UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

INTEGRACIÓN DE INFORMACIÓN AMBIENTAL EN ASISTENTES PERSONALES GESTIONADOS POR VOZ

Nazariy Gunko
Tutor: Pablo Varona Martínez

JULIO 2020

Integración de información ambiental en asistentes personales gestionados por voz

AUTOR: Nazariy Gunko
TUTOR: Pablo Varona Martínez

Grupo de Neurocomputación Biológica
Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2020

Resumen

Los asistentes personales son agentes software presentes en dispositivos inteligentes que pueden ayudar a las personas a llevar a cabo tareas de una manera cómoda e interactiva. Son una tecnología bastante reciente con mucho potencial para expandir sus aplicaciones a diferentes ámbitos de la vida de los seres humanos en el contexto del llamado Internet de las Cosas.

El objetivo de este Trabajo fin de Grado es la implementación de la funcionalidad necesaria para interconectar los asistentes personales de Alexa y Google Assistant con una nariz electrónica artificial. De esta forma, se desea poder obtener los datos e informes recogidos por la nariz y consultarlos mediante comandos de voz, que serán interpretados por los asistentes personales. También se podrán recibir alarmas por eventos específicos detectados en las señales obtenidas en la nariz.

La nariz mencionada anteriormente está formada por un conjunto de sensores que pueden detectar diferentes características ambientales, como pueden ser la presencia y cantidad de ciertas componentes orgánicas en el aire, por ejemplo, la cantidad de CO_2. El objetivo de la nariz es registrar las rutinas humanas y recoger toda la información ambiental relevante. En este TFG se abordan los aspectos de este proyecto relacionados únicamente con la obtención y comunicación de datos de la nariz por medio de asistentes personales.

Con la implementación de la conexión entre los asistentes personales y la nariz artificial, resulta posible añadir módulos que realicen un estudio más profundo de los datos que el que se utiliza para ilustrar este trabajo, creando una herramienta muy potente para la monitorización de la actividad humana en distintos entornos.

Palabras clave

Asistente personal, Alexa, Google Assistant, nariz electrónica artificial, monitorización de rutinas humanas.

Abstract

Intelligent personal assistants are software agents present in numerous intelligent devices that help people with different tasks making it easy and accessible by establishing a conversational interaction with people. This technology is quite recent and has room for improvement as it can be used in multiple different fields of human life in the context of Internet of Things.

The goal of this Bachelor Thesis is to interconnect the Alexa and Google Assistant personal assistants with an electronic nose device. This way, data collected by the electronic nose can be accessed via voice commands, and interpreted by the personal assistants, which will present the data with a voice answer. In addition to this functionality, the personal assistants will have the ability to notify the user with alarms from events detected in the nose.

The electronic nose mentioned in the last paragraph is a set of sensors that are able to detect multiple different aspects of the environment they are placed in, such as the presence of organic components like CO_2 in the air and their quantity. The purpose of the electronic nose is to noninvasively register human routines and collect environmental data. This Thesis only targets the gathering and communication of data part of this project.

Having implemented the communication between the electronic nose and the personal assistants, it is possible to integrate external modules that carry out deep data analysis, making this project a very powerful tool for the monitoring of human activity in different environments.

Keywords

Intelligent personal assistants, Alexa, Google Assistant, electronic nose, human routine monitoring.

Agradecimientos

En primer lugar, a mi familia por apoyarme todos estos años.

A mi tutor Pablo Varona Martínez por darme la oportunidad de llevar a cabo este trabajo.

A mis amigos por haberme soportado los días de estrés cercanos a la entrega del TFG.

INDICE DE CONTENIDOS

1 Introducción 1
1.1 Motivación 1
1.2 Objetivos 2
1.3 Organización de la memoria 2
2 Estado del arte 3
2.1 Sensorización 3
2.2 Asistentes personales 4
3 Diseño 7
3.1 Descripción general del problema 7
3.2 Requisitos funcionales 8
3.3 Asistente personal 8
3.4 La nariz electrónica y el almacenamiento de datos 10
3.5 El servidor 11
4 Desarrollo 12
4.1 Problemas 12
4.2 Implementación de las características del asistente 12
4.2.1 Google Assistant 12
4.2.2 Alexa 18
4.3 Implementación de la comunicación con el servidor 23
4.3.1 Google Assistant 23
4.3.1.1 Servicio web 23
4.3.1.2 Rutina de alarmas 23
4.3.2 Alexa 24
4.3.2.1 Servicio web 24
4.3.2.2 Rutina de alarmas 26
4.4 Implementación de la consulta de datos de la nariz 27
5 Integración, pruebas y resultados 28
5.1 Integración 28
5.2 Pruebas 28
5.3 Resultados 28
5.3.1 Resultados obtenidos para el asistente de Google 29
5.3.2 Resultados obtenidos para Alexa 32
6 Conclusiones y trabajo futuro 35
6.1 Conclusiones 35
6.2 Trabajo futuro 36
Referencias 37
Glosario 39
Anexos ..... - 1 -
A Manual de instalación ..... - 1 -

INDICE DE FIGURAS

FIGURA 2-1: FUNCIONAMIENTO SENSOR MOS [8] 3

FIGURA 2-2: ASISTENTE DE GOOGLE Y ALEXA 4

FIGURA 2-3: GOOGLE HOME MINI Y ALEXA ECHO DOT 5

FIGURA 3-1: ESQUEMA DE AGENTES 7

FIGURA 3-2: EJEMPLO DE PETICIÓN 9

FIGURA 3-3: EJEMPLO DE RESPUESTA 10

FIGURA 3-4: VISUALIZACIÓN DE DATOS DEL CLÚSTER CON LA SELECCIÓN DE TRES SENSORES Y SU EVOLUCIÓN EN EL TIEMPO 11

FIGURA 4-1: CONSOLA DE ACCIONES DE GOOGLE 13

FIGURA 4-2: DIALOGFLOW-INTENCIONES 14

FIGURA 4-3: ENUNCIADOS DE ENTRENAMIENTO 15

FIGURA 4-4: EJEMPLO DE PARÁMETRO: SENSOR 16

FIGURA 4-5: PARÁMETROS DE LA INTENCIÓN PRINCIPAL 16

FIGURA 4-6: PARÁMETROS Y FULFILLMENT 17

FIGURA 4-7: DEFINICIÓN DEL SERVICIO DE FULFILLMENT 18

FIGURA 4-8: CONSOLA DE DESARROLLO DE SKILLS DE ALEXA 19

FIGURA 4-9: NOMBRE DE INVOCACIÓN DE LA SKILL 19

FIGURA 4-10: PARÁMETROS DE LOS COMANDOS DE VOZ 20

FIGURA 4-11: DEFINICIÓN DEL SERVICIO DE FULFILLMENT 20

FIGURA 4-12: INTENCIONES DEFINIDAS PARA ALEXA 21

FIGURA 4-13: COMANDOS DE VOZ DE ENTRENAMIENTO DE LA INTENCIÓN PRINCIPAL EN ALEXA . 21

FIGURA 4-14: PARÁMETROS DE LA INTENCIÓN PRINCIPAL EN ALEXA 22

FIGURA 4-15: COMANDO PARA EJECUTAR LA INTENCIÓN EXTRA EN ALEXA 22

FIGURA 4-16: EJEMPLO DE CONTROLADOR DE INTENCIÓN EN UNA SKILL DE ALEXA 25

FIGURA 5-1: COMANDO INICIAL GOOGLE ASSISTANT 30

FIGURA 5-2: COMANDO CONSULTA DE ÚLTIMO DATO 30

FIGURA 5-3: CONSULTA DE MÁXIMOS DEL DÍA DE AYER Y DE LA SEMANA PASADA 31

FIGURA 5-4: CONSULTA DE VALORES MÁXIMOS DEL MES ACTUAL Y UN MES ESPECÍFICO 31

FIGURA 5-5: CONSULTA DE VALORES POR ENCIMA DE UN UMBRAL 32

FIGURA 5-6: INVOCACIÓN SKILL DE ALEXA 32

FIGURA 5-7: CONSULTAS DE ÚLTIMO DATO DE CADA SENSOR EN ALEXA 33

FIGURA 5-8: CONSULTA DE MÁXIMOS EN ALEXA 34

FIGURA 5-9: CONSULTA DE MÁXIMOS PARA PERIODOS DE TIEMPO GRANDES EN ALEXA 34

FIGURA 5-10: CONSULTA DE MÁXIMOS POR ENCIMA DE CIERTO UMBRAL EN ALEXA 35

INDICE DE TABLAS

TABLA 2-1: PRINCIPALES ALTAVOCES INTELIGENTES 6

TABLA 3-1: REQUISITOS FUNCIONALES 8

TABLA 4-1: PARÁMETROS DE CONSULTA 27

TABLA 5-1: DESCRIPCIÓN Y ENLACES A DEMOSTRACIONES DEL FUNCIONAMIENTO DE LOS ASISTENTES CON DISPOSITIVOS FÍSICOS 29

1 Introducción

1.1 Motivación

En las últimas décadas, las narices electrónicas han sido un campo de interés y desarrollo debido a su potencial para resolver problemas en un gran abanico de campos de la actividad humana, desde ámbitos científicos hasta comerciales. Desde la aparición del primer modelo inteligente de sensorización de gases por Persaud y Dodd en 1982 [1], se ha llevado a cabo investigación y desarrollo de este tipo de dispositivos principalmente en ámbitos comerciales e industriales [2]. Dichos dispositivos son capaces de simular el sistema olfativo de los seres vivos, para detectar olores y sus características principales.

El gran desarrollo de la Inteligencia Artificial en los últimos años ha sido una componente de gran importancia que ha fomentado el desarrollo y uso de las narices artificiales, al ser una herramienta muy potente para el procesado y análisis profundo de los datos obtenidos, siendo destacables los algoritmos de Aprendizaje Automático como pueden ser las redes neuronales artificiales [3-5].

Las aplicaciones de las narices electrónicas son cada vez más frecuentes, expandiéndose a nuevos campos. El proyecto para el cual se desarrolla este trabajo parte de la aplicación de las narices para la monitorización no invasiva de la actividad humana, principalmente en ambientes cognitivos, de estudio y aprendizaje, como pueden ser, por ejemplo, las propias clases de la UAM.

La segunda y principal componente principal de este TFG, son los asistentes personales, que poseen características interesantes para el proyecto.

Los asistentes personales son agentes software muy presentes en la actualidad en el contexto del Internet de las Cosas. Aunque a menudo pasan desapercibidos, en la actualidad todos los teléfonos móviles inteligentes cuentan con un asistente personal, correspondiente al desarrollado por el fabricante del dispositivo. En años recientes, también se han desarrollado dispositivos inteligentes específicamente diseñados para integrar al asistente personal y cada vez aparecen más aparatos compatibles que permiten automatizar diferentes tareas, principalmente en el ámbito del hogar.

La principal ventaja de los asistentes personales es su facilidad de uso, al poder una persona establecer una conversación con el dispositivo inteligente como si de una conversación con otro ser humano se tratara. Otro punto positivo, es que ofrecen a los desarrolladores la posibilidad de implementar funcionalidad personalizada para integrar al asistente en diferentes sistemas.

Sin embargo, como se describirá más adelante en el documento, debido a que los asistentes personales comerciales son una tecnología bastante reciente, presentan ciertos obstáculos para su integración.

1.2 Objetivos

El objetivo de este trabajo es implementar la funcionalidad necesaria para poder establecer una comunicación con un asistente personal por voz y mediante esta comunicación consultar datos recogidos por la nariz electrónica, a la vez que recibir alarmas automatizadas.

La comunicación debe ser bidireccional, una persona debe poder iniciar una conversación con el asistente y ejecutar de esta forma diferentes comandos para consultar datos de la nariz que se adquieren en tiempo real y quedan almacenados; pero, por otro lado, el asistente también debe ser capaz de iniciar una conversación, o simplemente comunicar a la persona una alarma de manera asíncrona. La alarma es generada por código ejecutado en el servidor, y el asistente debe notificar al usuario del suceso y las características de este.

En cuanto al procesado y análisis de los datos, se desea implementar un análisis muy simple, obteniendo valores máximos o valores máximos por encima de ciertos umbrales para varios sensores de ejemplo.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Estado del arte: se describirá brevemente la situación actual de las narices electrónicas y de los asistentes personales en la actualidad.
- Diseño: se expondrá una descripción a alto nivel de como se ha planteado el problema a resolver y de todas las componentes y agentes involucrados.
- Desarrollo: en este apartado se detalla toda la implementación.
- Integración, pruebas y resultados: se expondrán ejemplos de ejecución y funcionamiento de la funcionalidad implementada.
- Conclusiones y trabajo futuro: se analizará si se han alcanzado los objetivos y se expondrán posibles mejoras o funcionalidad a añadir al proyecto.

2 Estado del arte

2.1 Sensorización

En la actualidad, las narices electrónicas son usadas en multitud de ámbitos al ser una opción no intrusiva [2,6], a diferencia de otros dispositivos como las cámaras o los micrófonos, pudiendo controlar y monitorizar diferentes procesos sin interferir en el desarrollo de estos ni en aspectos relacionados con la privacidad.

Las narices electrónicas son especialmente frecuentes en diferentes procesos industriales. Un ejemplo de ello puede ser el estudio [7] que demuestra como se pueden detectar los diferentes niveles de maduración de los tomates mediante una nariz.

Una nariz a grandes rasgos es un conjunto de sensores, cada uno de los cuales se encarga de detectar componentes específicas del ambiente en el que se esté realizando el estudio. Dependiendo del tipo de ambiente, se utilizan sensores de diferente arquitectura y funcionamiento. En el ámbito de detección de aroma, los sensores más utilizados son los semiconductores de metal oxidado (MOS) y los transistores de efecto de campo metal-óxido-semiconductor (MOSFET).

En la Figura 2-1 se puede observar un esquema del funcionamiento de un sensor MOS. Con la aparición del gas reductor, los electrones donados por el oxígeno se desprenden hacia el dióxido de estaño, reduciendo la resistencia del conductor y permitiendo que haya corriente en el sensor [8].

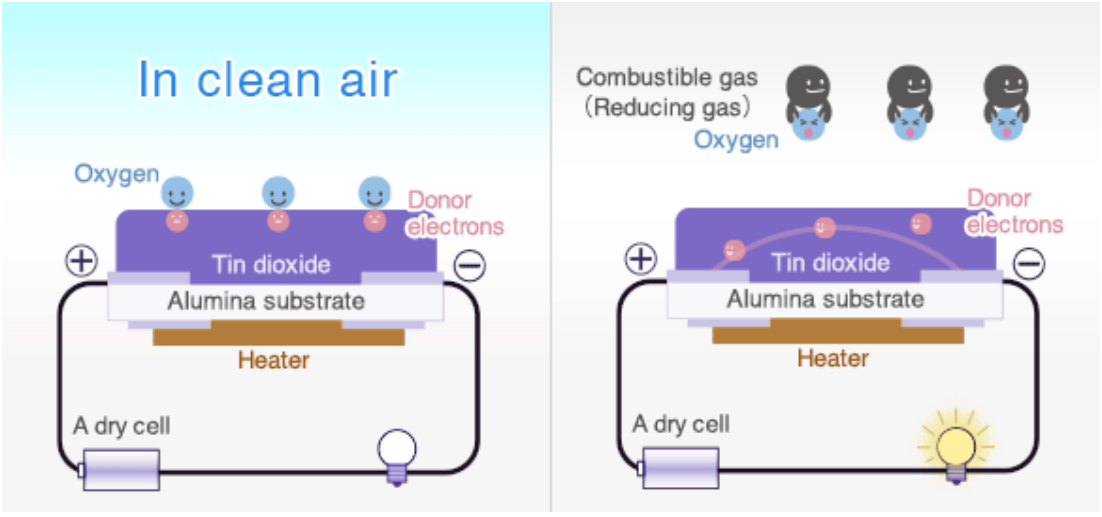

Figura 2-1: Funcionamiento sensor MOS [8]

Estudios recientes [9,10] exponen la necesidad de aplicar técnicas para mejorar las cualidades de los sensores mencionados anteriormente, a la hora de discriminar las sustancias odorantes. Esto es debido a que los sensores no son invariables al tiempo, por lo cual dependiendo de las sustancias con las que el sensor haya estado en contacto en el pasado, puede influir en el resultado obtenido en la actualidad. Estos estudios proponen distintas estrategias para compensar la deriva inherente a los sensores de manera dinámica y flexible.

2.2 Asistentes personales

Los asistentes personales son una tecnología muy reciente, apareciendo los primeros en la década del 2010. A pesar de ello, se han integrado rápidamente en nuestras vidas como veremos más adelante.

En la actualidad, los asistentes personales más importantes son Alexa de Amazon, y Assistant de Google, como indican estudios sobre participación de mercado de altavoces inteligentes que integran asistentes personales [11], según el cual sus respectivos altavoces inteligentes suman entre ambos un 68% de participación de mercado en 2018 en Estados Unidos. También cabe destacar el asistente de Apple, Siri, cuyo uso está muy extendido pero más limitado que los anteriores debido a que sólo está presente en dispositivos Apple.

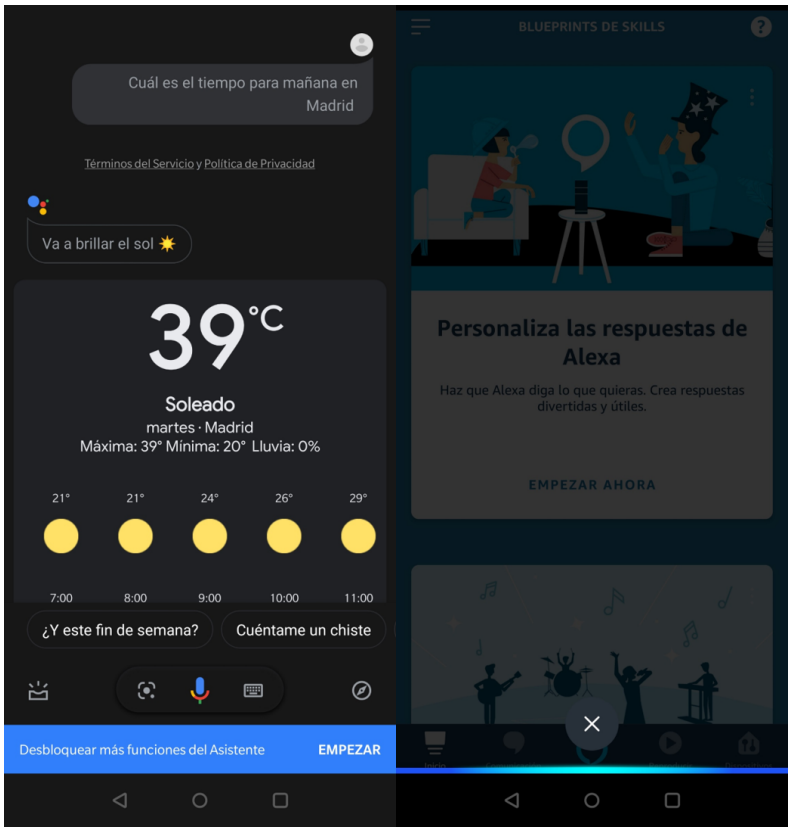

Figura 2-2: Asistente de Google y Alexa

Un asistente personal es un programa software que, utilizando algoritmos de inteligencia artificial como el Procesado de Lenguaje Natural (NLP), es capaz de procesar la voz humana, extrayendo información para posteriormente realizar algún tipo de tarea y presentar los resultados de esta en forma de voz humana [12]. De esta forma, los asistentes personales simulan una conversación bidireccional entre seres humanos, para así poder automatizar y hacer más cómodo el llevar a cabo ciertas tareas. Las tareas que los asistentes personales pueden llevar a cabo son cada vez más variadas, desde consultas simples como puede ser consultar la hora o el tiempo que va a hacer mañana, que sustituirían una simple búsqueda en un navegador, hasta tareas dentro de ciertas aplicaciones compatibles con las que el asistente está integrado, como puede ser la aplicación de música Spotify o la plataforma de contenido audiovisual Netflix.

No solo las tareas que los asistentes pueden llevar a cabo se están expandiendo, sino que también cada vez más dispositivos integran la funcionalidad para mejorar la experiencia del usuario. Un ejemplo de esto puede ser el sector del automóvil. En la actualidad, la mayoría de las grandes marcas desarrollan automóviles con sistemas de info-entretenimiento o control avanzados que se basan en los sistemas operativos de Android o IOS, los cuales por defecto traen integrados los asistentes de Google y Siri respectivamente. Algunos fabricantes desarrollan sus propios sistemas de control por voz inteligentes, como puede ser el sistema MBUX de Mercedes-Benz, que permiten controlar diferentes características del automóvil como puede ser la calefacción o las luces ambientales. El uso de asistentes personales en el sector del automóvil es un claro ejemplo de la utilidad y las mejoras que trae un modelo conversacional, al permitir al usuario realizar acciones sin apartar la vista y atención de la carretera, favoreciendo de esta forma su seguridad.

A pesar de la expansión de los asistentes personales a diferentes campos como se ha expuesto en el párrafo anterior, actualmente los dispositivos con asistentes más utilizados son los altavoces inteligentes y los smartphones o teléfonos móviles inteligentes. En la Figura 2-3 se pueden ver como ejemplo dos altavoces inteligentes que integran los asistentes de Google y Alexa respectivamente. Estos dispositivos integran micrófonos y altavoces de alta calidad.

Figura 2-3: Google Home Mini y Alexa Echo Dot

En la siguiente Tabla 2-1 se recopilan los principales altavoces inteligentes del mercado junto con algunas de sus principales características.

Nombre	Asistente	Pantalla	Precio *	Foto
Home Mini	Google Assistant	No	59€	
Nest Hub	Google Assistant	Si	89,99€	
Echo Dot	Alexa	No	39,99€	
Echo Show 5	Alexa	Si	69,99€	
HomePod	Siri	No	329€	

Tabla 2-1: Principales altavoces inteligentes

* Los precios reflejados corresponden a los anunciados en las respectivas páginas oficiales de los fabricantes.

3 Diseño

En este apartado se detalla la arquitectura del problema y la solución a alto nivel diseñada para alcanzar los objetivos planteados, entrando al detalle de la implementación en el apartado de Desarrollo.

3.1 Descripción general del problema

El objetivo es conseguir enlazar todos los agentes para permitir al usuario recuperar los datos recogidos por la nariz electrónica mediante la interacción con el asistente personal. El usuario debe poder iniciar una conversación de voz con el asistente, el cual le responderá con los datos solicitados. Pero también, un requisito es que el propio asistente debe ser capaz de lanzar una alarma cuando se detecte algún evento.

En la Figura 3-1 se muestra un esquema de los agentes involucrados y la comunicación entre los mismos.

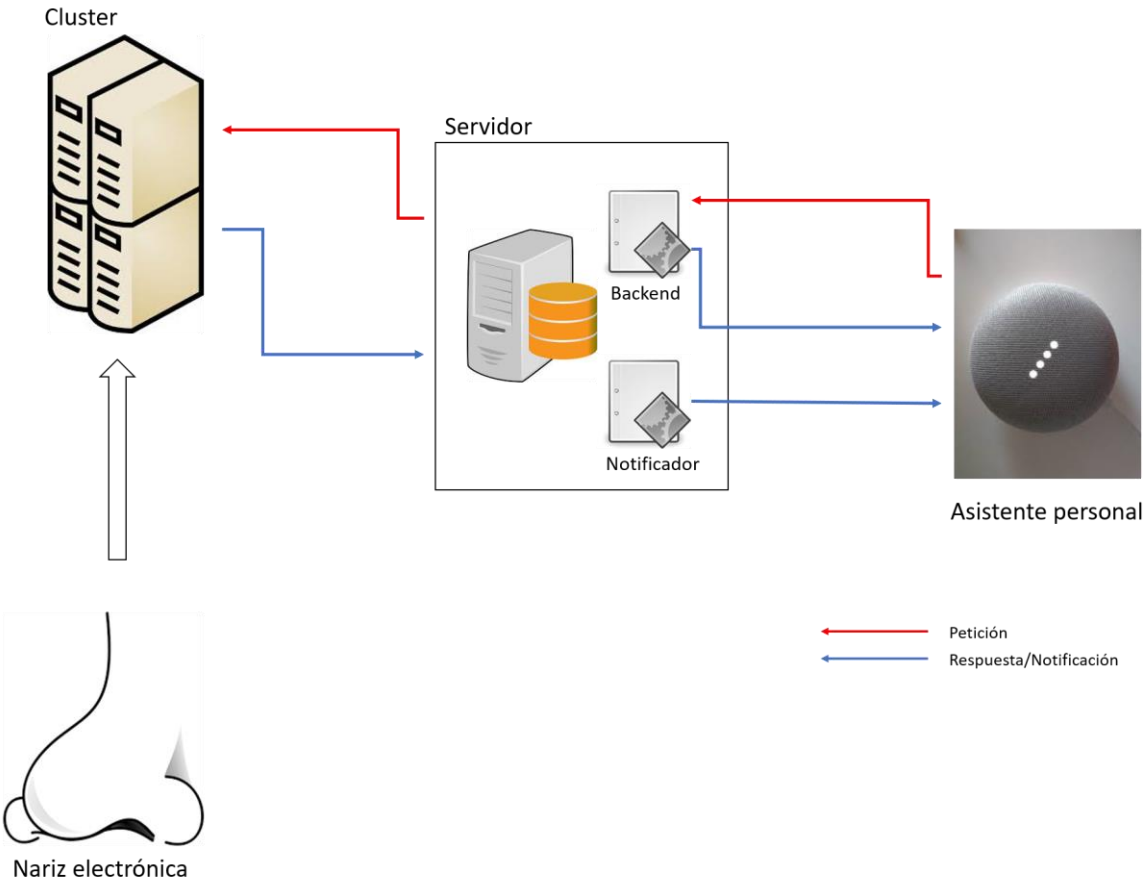

Figura 3-1: Esquema de agentes

3.2 Requisitos funcionales

En la Tabla 3-1 se listan los requisitos funcionales del problema planteado que se quiere resolver.

Requisito	Descripción
RF-1	El asistente debe ser capaz de entender comandos de voz relativos a la comunicación de datos obtenidos por la nariz.
RF-2	Se podrá consultar datos para un sensor específico.
RF-3	Se podrá consultar datos para un rango temporal específico.
RF-4	Se realizará un procesado de los datos simple para obtener valores máximos.
RF-5	El asistente notificará de eventos en la nariz mediante alarmas de forma automática.

Tabla 3-1: Requisitos funcionales

3.3 Asistente personal

El asistente personal será el input y output del proceso. Será el encargado de recibir los comandos de voz y de responder a ellos con una respuesta de voz también. El primer comando será siempre activar el dispositivo con la palabra clave, por ejemplo "*Alexa*" u "*OK, Google*", y a continuación indicar que queremos ejecutar nuestra aplicación con "*ejecuta ...*". Tras lo anterior, ya podemos decirle al asistente el comando que queremos ejecutar.

Una vez que el asistente recibe el comando, lo interpreta aplicando algoritmos de NLU para extraer información y en base a ello construye una petición. La información imprescindible es:

- Primero, la acción que se quiere llevar a cabo, que se traducirá en una *intención* dentro de nuestra implementación.
- Segundo, los parámetros con los que se quiere ejecutar la acción. Las acciones pueden no tener parámetros, o tener varios parámetros definidos de los cuales el algoritmo debe obtener el valor.

Tras obtener la información, se construye una petición que será enviada al servicio en el servidor. Aunque los detalles de la petición dependen de la implementación y de la plataforma, en la mayoría de los casos el cuerpo de la petición consiste en un diccionario de pares clave-valor en formato JSON, que es enviado dentro de una petición POST HTTP. La Figura 3-2 muestra un ejemplo de cabecera y cuerpo enviados en una petición creada por Alexa.

```
POST / HTTP/1.1
Content-Type : application/json;charset=UTF-8
Host : your.application.endpoint
Content-Length :
Accept : application/json
Accept-Charset : utf-8
```

```
{
  "version": "1.0",
  "session": {
    "new": true,
    "sessionId": "amzn1.echo-api.session.[unique-value-here]",
    "application": {
      "applicationId": "amzn1.ask.skill.[unique-value-here]"
    },
    "attributes": {
      "key": "string value"
    },
    "user": {
      "userId": "amzn1.ask.account.[unique-value-here]",
      "accessToken": "Atza|AAAAAAAA...",
      "permissions": {
        "consentToken": "ZZZZZZZ..."
      }
    }
  },
```

Figura 3-2: Ejemplo de petición

El objeto más interesante del cuerpo de la petición en nuestro caso será el objeto bajo la clave "*request*", que contendrá los datos extraídos del comando de voz, como se explicó anteriormente.

Tras realizar la petición, el asistente se bloquea esperando una respuesta del servidor durante un máximo de segundos 8 segundos en Alexa y 10 segundos en Google, transcurridos los cuales se notificará un error en caso de que no se haya recibido respuesta. La respuesta que espera el asistente sigue el mismo formato que la petición, en cuanto a que utiliza el protocolo HTTP y espera un objeto JSON en el cuerpo de la petición. La Figura 3-3 muestra un ejemplo de respuesta esperada por el asistente.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length:

{
  "version": "string",
  "sessionAttributes": {
    "key": "value"
  },
  "response": {
    "outputSpeech": {
      "type": "PlainText",
      "text": "Plain text string to speak",
      "playBehavior": "REPLACE_ENQUEUED"
    },
```

Figura 3-3: Ejemplo de respuesta

Dentro de la respuesta, el objeto más interesante es el diccionario bajo la clave "*response*", que modelará la respuesta del asistente. Las respuestas pueden variar desde una simple respuesta de voz, hasta una tarjeta personalizada que podrá ser mostrada al usuario en dispositivos que cuenten con pantalla.

El asistente también debe ser capaz de notificar al usuario de manera asíncrona, es decir, sin que el usuario haya iniciado una conversación con el asistente, en caso de que ocurra algún evento que constituya una alarma, por ejemplo, que un sensor supere un valor por encima de un umbral establecido. Este requisito va en contra de algunos principios generales de los asistentes personales, y la única manera de conseguir dicho comportamiento es mediante la implementación de Notificaciones, como se detalla en el apartado de Desarrollo.

3.4 La nariz electrónica y el almacenamiento de datos

La nariz electrónica es la fuente de datos de nuestro esquema. Esta nariz registra la evolución en el tiempo de la señal obtenida con sensores de CO_2, aminas, distintas sustancias orgánicas y alcoholes, temperatura, humedad, etc. Los datos recogidos por la nariz utilizada en este proyecto son enviados al Clúster del Grupo de Neurocomputación Biológica de la UAM donde son almacenados. Gracias a esto se pueden consultar datos tanto en tiempo real (con cierto retardo de unos segundos) como datos más antiguos. En la Figura 3-4 se puede observar una visualización de los datos almacenados en el clúster mediante el servicio eNose desarrollado para este propósito.

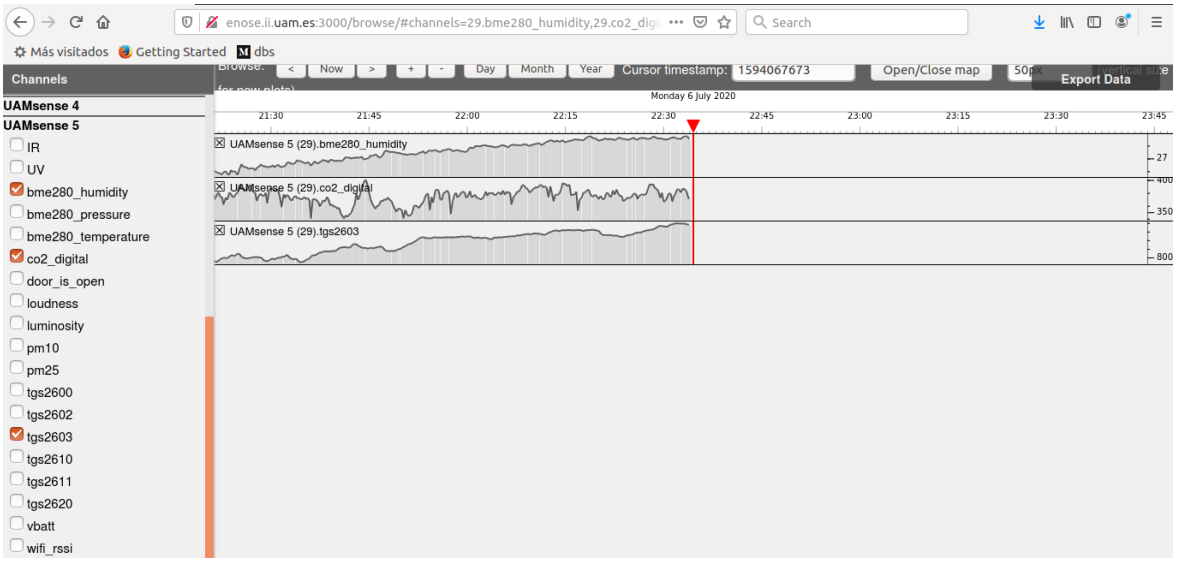

Figura 3-4: Visualización de datos del Clúster con la selección de tres sensores y su evolución en el tiempo

Como se puede observar es posible consultar datos específicos para unos sensores concretos en un rango de tiempos. Sin embargo, para nuestra aplicación, vamos a necesitar obtener dichos datos de manera automática, desde nuestro código. Para ello, se cuenta con un servicio desplegado en la URL: *http://enose.ii.uam.es:3000/api/v1/feeds/* , la cual acepta peticiones GET HTTP con los parámetros de la consulta concatenados a la URL anterior. Gracias a esto, nuestro código del servidor es capaz de obtener datos para presentárselos al usuario, no sin antes realizar un procesado de estos.

3.5 El servidor

Nuestro servidor será básicamente el enlace entre el asistente y la nariz.

El servidor será formado por dos rutinas principales:

- Un servicio web que es el encargado de recibir las peticiones del asistente, formar la petición correspondiente para consultar datos del clúster, y formar una respuesta con los datos obtenidos que será devuelta al asistente. Los asistentes de Google o Amazon permiten al desarrollador desplegar y utilizar una aplicación web propia, las cuales pueden ser creadas fácilmente con frameworks como Flask, o utilizar plataformas como pueden ser las Cloud Functions de Google, o Lambda de AWS. Utilizar un servicio web propio le proporciona al desarrollador mayor libertad, al poder definir todos los parámetros de la plataforma, como puede ser por ejemplo el lenguaje de programación. Sin embargo, las APIs de desarrollo de acciones personalizadas pueden no soportar el lenguaje de programación escogido, por lo cual no se podrá hacer uso de dicha funcionalidad. En el apartado de desarrollo se expondrán distintas implementaciones usando tanto las plataformas proporcionadas como servicios propios.

- La segunda rutina consiste básicamente en un script que estará en continua ejecución, lanzando peticiones al Clúster cada cierto periodo de tiempo para obtener los datos más actuales registrados por la nariz. El objetivo de esta rutina es detectar cualquier dato que sea peligroso o indeseado, como puede ser un nivel de

CO_2 demasiado elevado en un aula, para notificárselo al usuario y tomar las acciones correspondientes. Para ello, la rutina enviará una petición asíncrona al asistente en forma de notificación, ya que las notificaciones o eventos proactivos son la única forma de conseguir este comportamiento.

4 Desarrollo

En esta sección se detallará paso a paso todo el procedimiento de implementación del sistema de comunicación descrito en el apartado de diseño. La comunicación ha sido implementada para los asistentes personales de Google y Alexa. Se describirán todos los pasos para ambos asistentes y se expondrá como han afectado las restricciones descritas en el apartado 4.1 Problemas al desarrollo, y los caminos alternativos que se han tomado para poder conseguir el comportamiento deseado.

4.1 Problemas

Una vez expuesto el diseño deseado, cabe destacar algunos inconvenientes que presentan las herramientas para el desarrollo de acciones personalizadas para los asistentes personales. Dichos inconvenientes forzaron tomar caminos y soluciones alternativas a la hora de desarrollar el sistema. Los principales inconvenientes encontrados son los siguientes:

- La plataforma de Google para desarrollar la lógica de servidor descrita en el apartado anterior sólo admite los lenguajes de Java, Kotlin y Node.js.

- El sistema de notificaciones del Asistente de Google limita el número de notificaciones que una aplicación puede enviar al usuario a 10 al día, lo cual es una restricción fuerte a la rutina de alertas.

- Con el objetivo de que la interacción con el asistente sea lo más parecido a una conversación humana, el tiempo máximo que el servidor tiene para enviar una respuesta está muy limitado, siendo de 8 segundos en el caso de Alexa.

4.2 Implementación de las características del asistente

A continuación, se detallan los pasos seguidos para conseguir crear acciones personalizadas, conseguir que el asistente entienda los comandos de voz deseados y que el asistente genere las peticiones que serán enviadas a nuestro servicio de back end de manera correcta. El proceso es bastante similar para ambos Alexa y asistente de Google. Se detallará cada uno de ellos por separado.

4.2.1 Google Assistant

El primer paso es crear un proyecto en la consola de acciones de Google [13]. Una vez tenemos el proyecto creado, es necesario añadir un nombre de invocación para comunicar al asistente que vamos a querer ejecutar comandos del marco de nuestra acción personalizada.

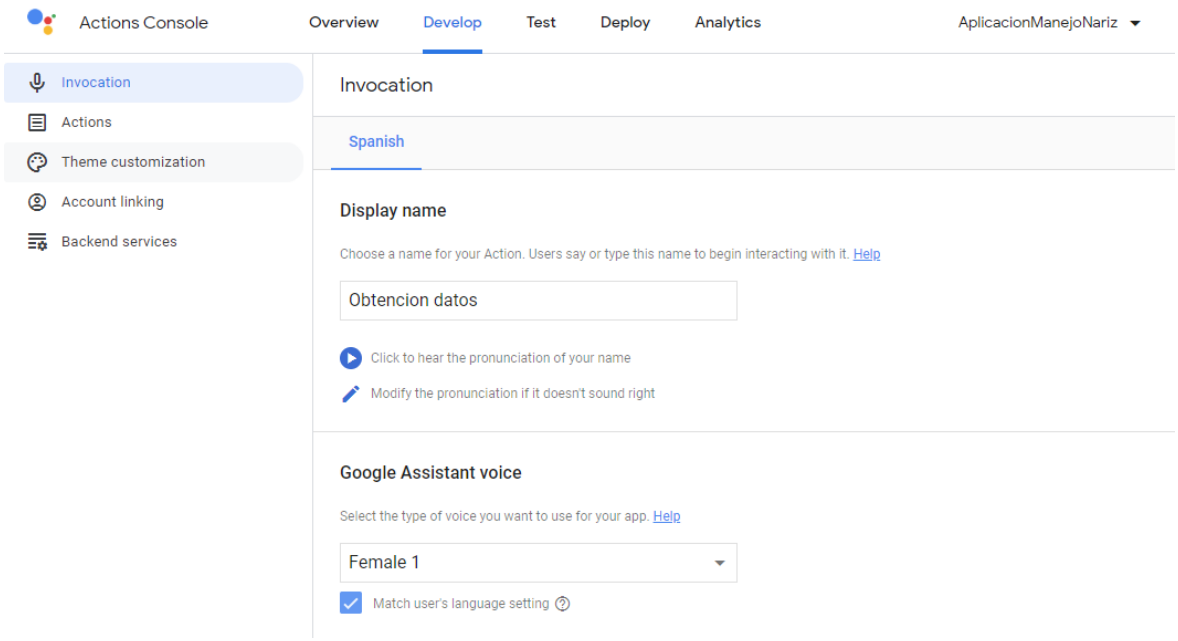

Figura 4-1: Consola de Acciones de Google

Una vez asignado un nombre de invocación, como se puede observar en la Figura 4-1, podremos invocar nuestra aplicación con un comando de voz como "*OK Google, ejecuta Obtención Datos*".

Tras esto, añadimos una acción y pasamos a personalizarla con la herramienta de Dialogflow. En Dialogflow podremos desarrollar todos los aspectos que harán que nuestro asistente sea capaz de entender nuestros comandos de voz y poder comunicarse con nuestro back end. En la Figura 4-2, en el menú de la izquierda, se pueden observar los diferentes aspectos que podemos modificar de nuestra acción. Nos centraremos sólo en los más importantes.

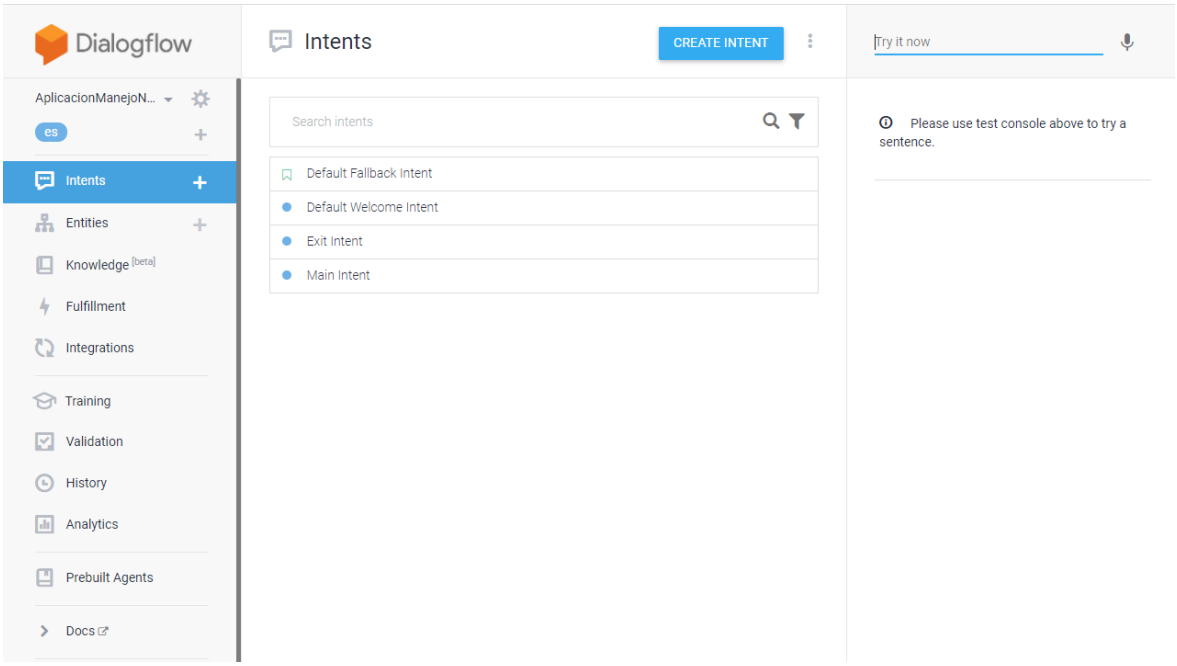

Figura 4-2: Dialogflow-Intenciones

En el apartado de Intents, definiremos las intenciones, o tareas específicas que el usuario quiere llevar a cabo. En nuestro caso hemos creado tres intenciones, una de bienvenida para cuando el usuario ejecuta nuestra aplicación, la intención está directamente asignada al evento de “Welcome”; una de salida para cuando el usuario quiere dejar de ejecutar comandos de nuestra aplicación, la cual tiene una respuesta estática asignada y no pasa por el back end; y la más importante, la intención principal, que será la que procese los comandos de obtención de datos.

Centrándonos en la Main Intent, lo primero que hacemos es añadir los comandos que va a soportar nuestra intención, con distintas variaciones para que el entrenamiento del algoritmo de NLU sea más preciso. La Figura 4-3 muestra los enunciados de entrenamiento utilizados para nuestra Main Intent.

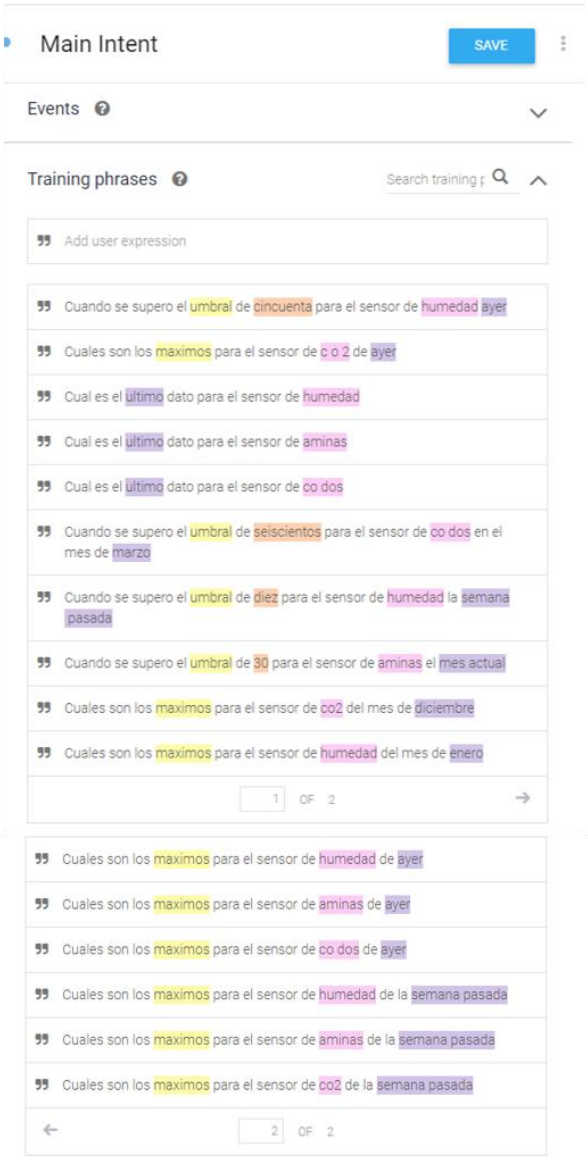

Figura 4-3: Enunciados de entrenamiento

Las palabras resaltadas en distintos colores son los parámetros que el algoritmo de NLU va a detectar y que van a ser enviados a nuestro servicio de back end. Dichos parámetros se definen en el apartado Entities, como se puede observar en la Figura 4-4 con un ejemplo de definición del parámetro Sensor. Se definen los posibles valores que puede tomar el parámetro y los sinónimos de cada valor.

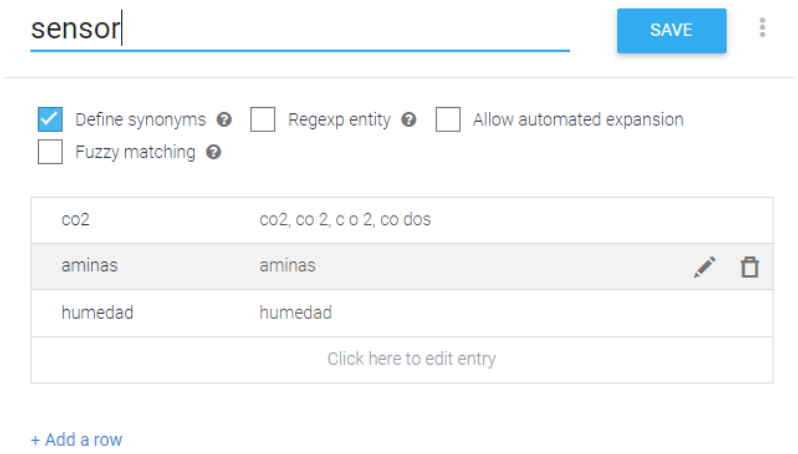

Figura 4-4: Ejemplo de parámetro: Sensor

Los parámetros necesarios para nuestros comandos se pueden en la Figura 4-5, que básicamente son los que el servidor necesita conocer para poder extraer los datos deseados del clúster. Los tres primeros parámetros son personalizados, mientras que el cuarto es uno básico definido por defecto en Dialogflow, que va a corresponder al umbral por encima del cual se quiera realizar la consulta.

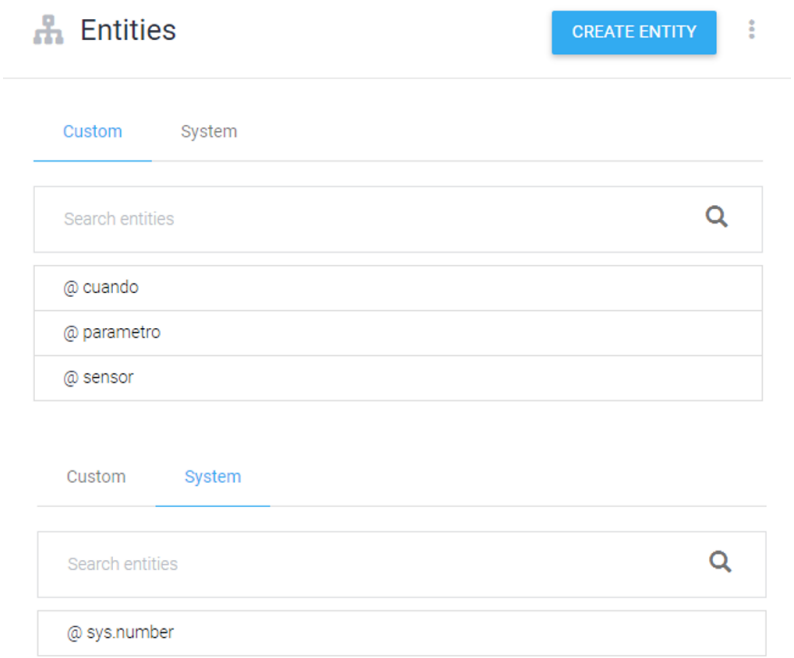

Figura 4-5: Parámetros de la intención principal

El último paso por hacer dentro de nuestra intención principal es añadir los parámetros definidos anteriormente, dándoles un nombre mediante el cual podremos acceder a ellos en el objeto JSON de la petición que le llegará a nuestro servicio y definiendo si son Required, es decir, si dichos parámetros no están presentes en el comando, se definirá un

Reprompt, o lo que es lo mismo, una frase que preguntará al usuario el valor del parámetro que falta. También será necesario activar el Webhook Fulfillment. Ver Figura 4-6.

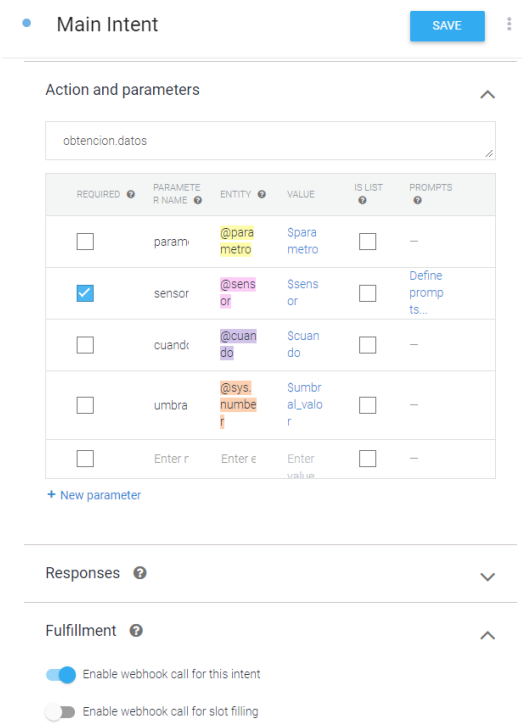

Figura 4-6: Parámetros y Fulfillment

Al activar el Webhook Fulfillment, permitimos que la intención sea procesada por el servicio que se encuentra desplegado en la URL que se indica en el apartado Fulfillment, como se puede ver en la Figura 4-7.

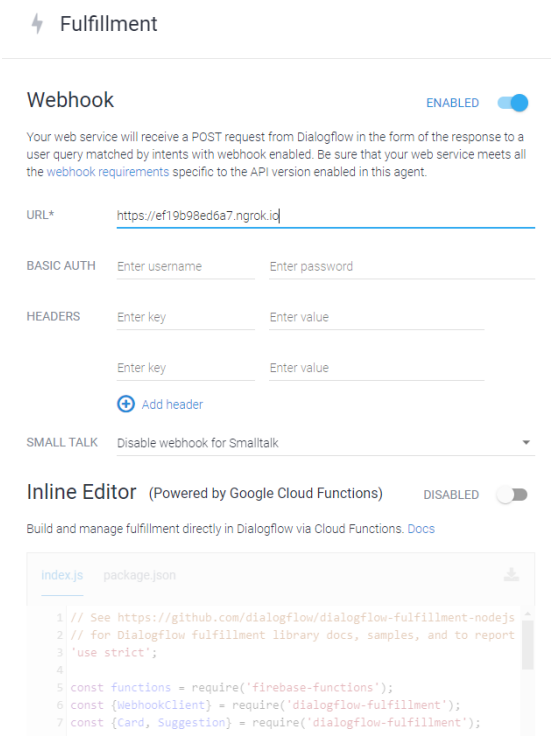

Figura 4-7: Definición del servicio de Fulfillment

Como se puede observar en la Figura 4-7, existen dos opciones para implementar el servicio web:

- Webhook: se añade la URL pública accesible por Google en la cual se encuentra nuestro propio servicio desplegado por nosotros. Es la opción elegida en nuestro caso debido a que, como se explicó en el Apartado 3-5, no se puede utilizar Python para desarrollar nuestra funcionalidad en la plataforma sugerida por Google.

- Google Cloud Functions: plataforma sugerida por Google, en caso de seleccionar la cual, Google se encargará de alojar el servicio web en la nube evitando al desarrollador tener que configurar la arquitectura.

Tras realizar los pasos descritos en este apartado, ya tendríamos configurado nuestro agente para interpretar los comandos de voz y comunicarse con nuestro back end.

4.2.2 Alexa

El procedimiento de implementación para Alexa es muy similar al del asistente de Google, en este caso utilizando las herramientas proporcionadas por Amazon.

En Alexa, las acciones personalizadas se denominan habilidades o Skills. Para crear una Skill, es necesario acceder a consola de desarrollo de Alexa [14]. Tenemos creada la Skill como se puede observar en la Figura 4-8, y a continuación pasamos a observar cómo está implementada.

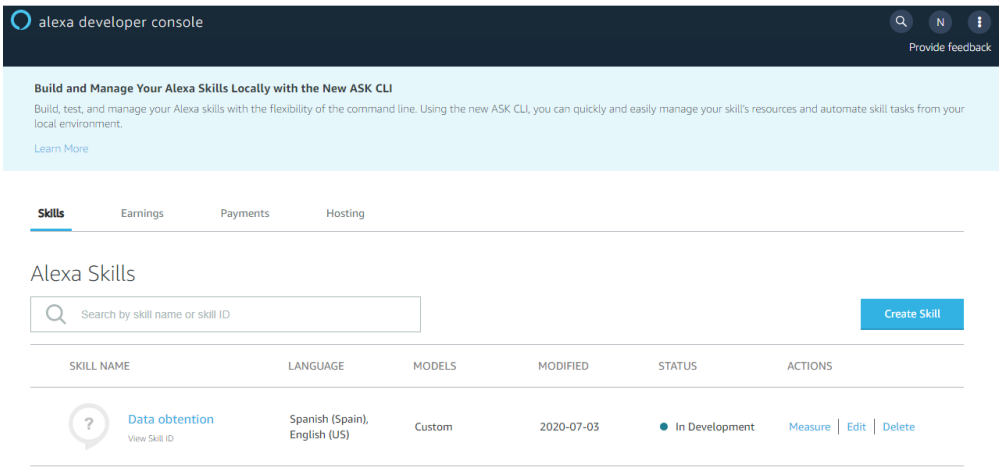

Figura 4-8: Consola de desarrollo de Skills de Alexa

Debido a que la implementación de la Skill es muy similar a la implementación de la acción para el asistente de Google ya descrita, simplemente se muestran a modo ilustrativo las Figuras 4-9:4-15 sin entrar en excesivo detalle.

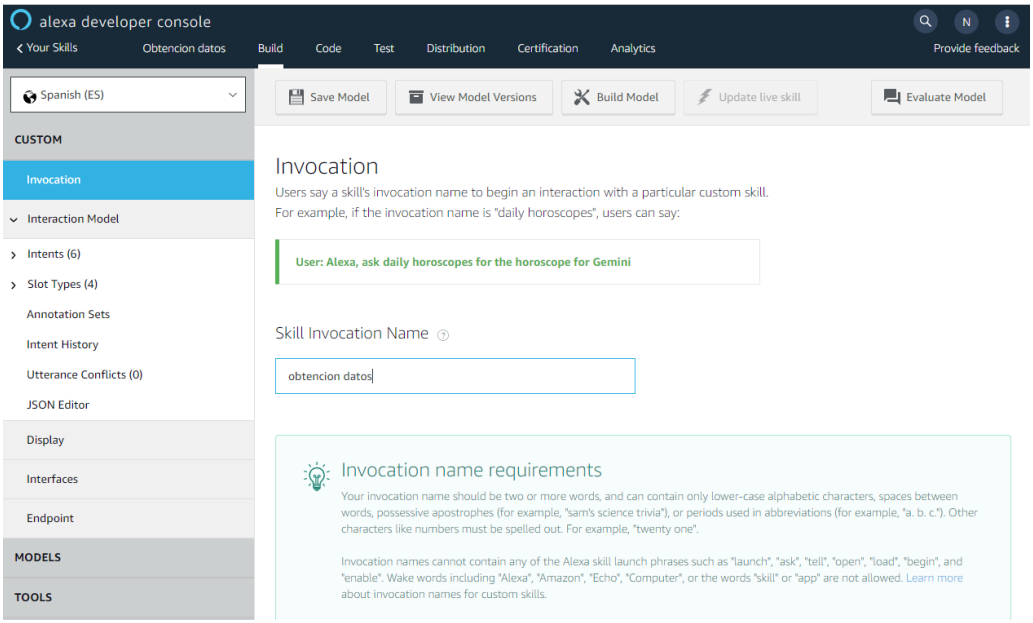

Figura 4-9: Nombre de invocación de la Skill

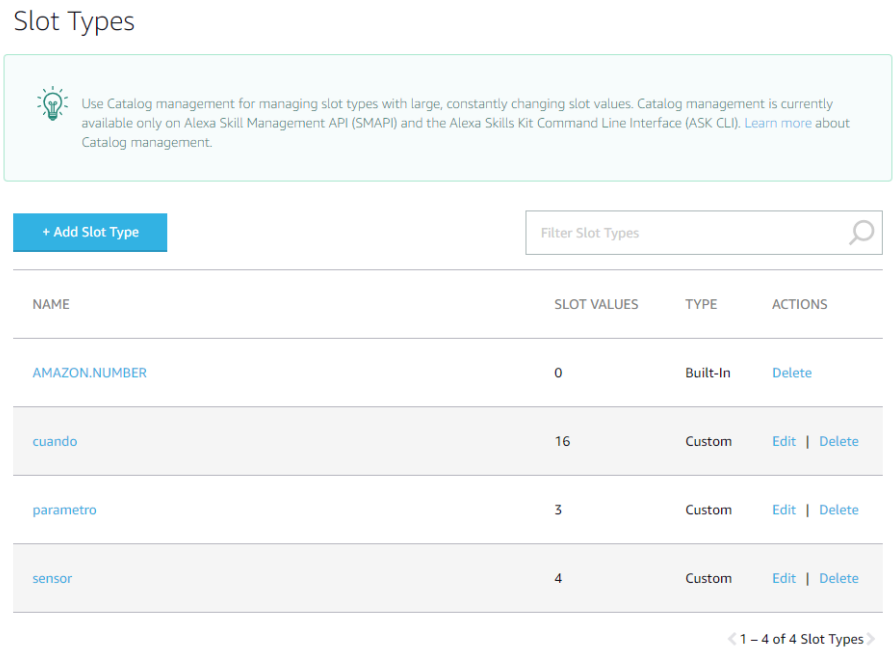

Figura 4-10: Parámetros de los comandos de voz

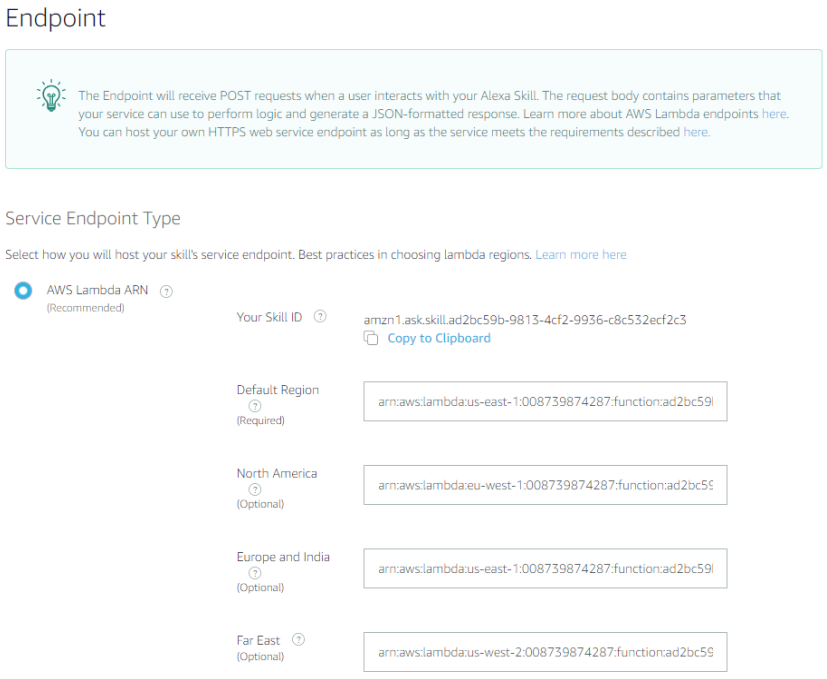

Figura 4-11: Definición del servicio de Fulfillment

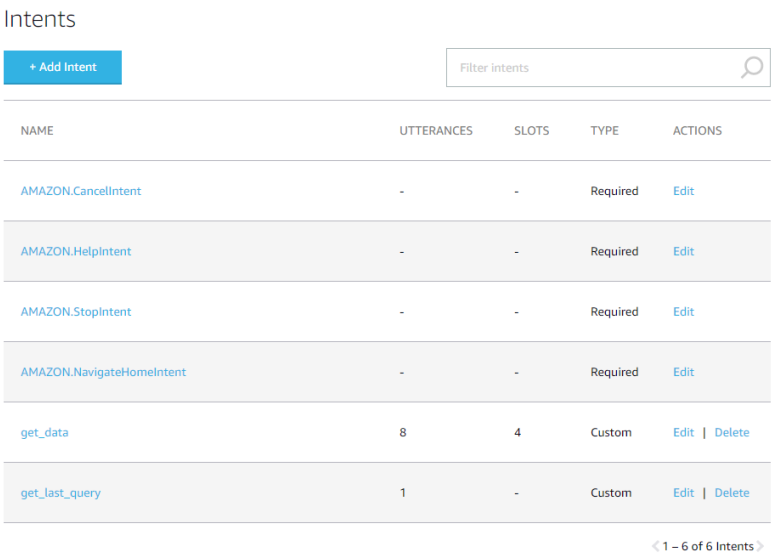

Figura 4-12: Intenciones definidas para Alexa

Figura 4-13: Comandos de voz de entrenamiento de la intención principal en Alexa

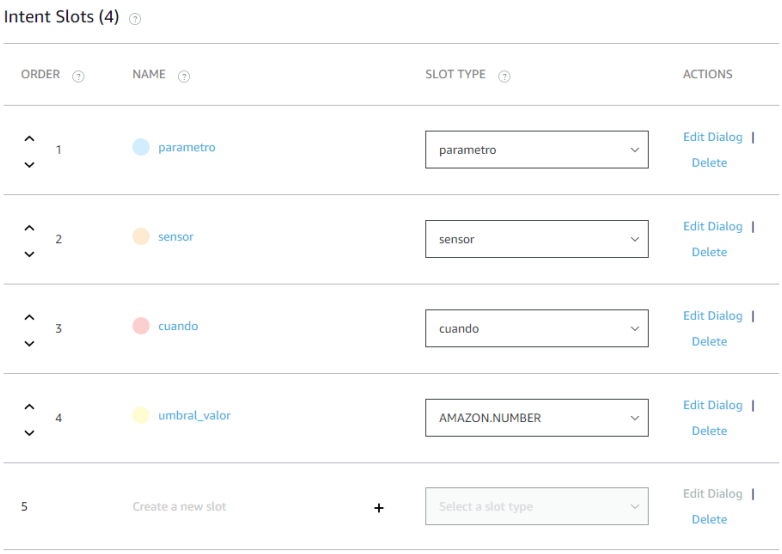

Figura 4-14: Parámetros de la intención principal en Alexa

Figura 4-15: Comando para ejecutar la intención extra en Alexa

Sí que cabe destacar algunos detalles distintos de la implementación para Alexa respecto a la del asistente de Google.

Como se puede observar en la Figura 4-11, el dominio donde se encuentra desplegado el servicio web de Fulfillment es un dominio del servicio Lambda de AWS. Se ha optado por alojar el servicio en la plataforma Lambda, ya que en Alexa sí que es posible desarrollar el servicio de back end utilizando Python, ya que existe una API específica para este lenguaje. Con ello, también se podrá observar más adelante las diferencias entre ambas implementaciones.

En las Figuras 4-12 y 4-15 podemos observar que se ha creado una intención extra llamada "get_last_query". La explicación del porqué de esta intención extra se detallará en el Apartado 4.2.2.

4.3 Implementación de la comunicación con el servidor

En este apartado se detallará la implementación de la comunicación con el servidor de nuestro diseño. La comunicación está implementada por dos rutinas principales, como se describió en el apartado de diseño, un servicio web encargado de enlazar al asistente con la nariz, y una rutina en constante ejecución encargada de generar las alarmas. A continuación, se presentan las implementaciones de dichas rutinas tanto para el asistente de Google como para Alexa.

4.3.1 Google Assistant

4.3.1.1 Servicio web

Debido a la restricción de que la API de Google y la plataforma Google Cloud Functions no admiten actualmente el uso del lenguaje de programación Python para el desarrollo del servicio de back end que va a atender las peticiones del asistente, se ha optado por desarrollar y desplegar un servicio web propio, utilizando el framework de Flask para ello. El servicio debe ser accesible por el servidor de Google, para lo cual es necesario que nuestro servicio esté desplegado en una URL pública, la cual se indicará en la Intent como se puede observar en la Figura 4-7. Para desplegar nuestro servicio, se ha utilizado la herramienta Ngrok, que permite crear un túnel hacia el puerto de localhost donde está corriendo nuestra aplicación de Flask.

La funcionalidad de nuestro servicio web es bastante sencilla, y se puede resumir en los siguientes puntos:

- La aplicación recibe peticiones POST como la que se puede observar en la Figura 3-2. Del cuerpo de la petición, se obtienen los valores que el asistente ha extraído para los diferentes parámetros de los comandos de voz.

- En función de dichos parámetros, se realiza una petición a la API de obtención de datos de la nariz.

- Con los datos obtenidos, se genera una respuesta que contiene el texto de la respuesta, que el asistente reproducirá con un mensaje de voz para el usuario.

4.3.1.2 Rutina de alarmas

Al igual que el servicio web, la implementación de la funcionalidad de alarmas para el asistente de Google también ha sido afectada por una de las restricciones señaladas en el apartado 4-1 Problemas. En concreto, la restricción que limita el uso de notificaciones a 10 al día como máximo, ha forzado tomar un camino alternativo para conseguir el comportamiento deseado.

Un altavoz inteligente como el Google Home Mini, es un dispositivo compatible con Google Chromecast, lo cual permite que otro dispositivo conectado a la misma red Wifi local pueda enviar audio al altavoz para que éste lo reproduzca. La tecnología es la misma que se utiliza, por ejemplo, para reproducir un video de Youtube en un televisor desde un teléfono móvil. Esta característica del altavoz nos permite enviar un texto de alarma que será reproducido por el altavoz de forma asíncrona, consiguiendo un comportamiento similar al de las notificaciones y, además, sin la restricción de 10 notificaciones al día.

La librería de Python *PyChromecast* [15] nos proporciona la funcionalidad necesaria para establecer comunicación con el altavoz y enviar el archivo de audio para su reproducción. Tras detectar la alarma y generar el texto correspondiente, es necesario convertir dicho texto a audio y almacenarlo en un fichero .mp3. Para ello, hacemos uso de la librería *gTTS* (Google Text-to-Speech) [16], que nos permite generar un archivo de audio en formato .mp3.

De esta forma, conseguimos el comportamiento de alarmas deseado. Sin embargo, una desventaja de dicha implementación es que el altavoz debe encontrarse en la misma red local que el dispositivo que este ejecutando la rutina de alarma. En caso de que se desee ejecutar la rutina de alarma en un servidor remoto, sería necesario tener, por ejemplo, una Raspberry ejecutando la funcionalidad de Chromecast, y comunicar con la misma desde el servidor remoto cuando se quiera lanzar una alarma.

4.3.2 Alexa

4.3.2.1 Servicio web

Para implementar la funcionalidad de servidor en el caso de Alexa, se ha optado por utilizar la API para Python: *ASK SDK for Python*. Además, el servicio web se ha alojado en la plataforma Lambda de Amazon. La plataforma Lambda, genera un nivel de abstracción para el desarrollador, haciendo que éste no se tenga que preocupar del despliegue o la arquitectura del servicio, pudiendo centrarse en el desarrollo del código de la funcionalidad. Además de lo mencionado anteriormente, se proporciona automáticamente un *bucket* en el servicio de almacenamiento en la nube S3 de Amazon, para poder almacenar cómodamente los archivos multimedia de nuestra Skill. También, nuestro código se integra de manera automática en el servicio CloudWatch de Amazon, el cual permite al desarrollador consultar las trazas de ejecución de la funcionalidad.

A continuación, vamos a observar un ejemplo de uso de la API *ASK SDK*. En la Figura 4-16 podemos ver la implementación del controlador de la Intent de bienvenida de nuestra Skill. El controlador de la Intent principal de nuestra Skill, “get_data”, sigue el mismo formato, con la diferencia de que su respuesta no es estática, sino que en función de los parámetros extraídos del objeto “handler_input” se realiza una petición a la API de obtención de datos de la nariz.

```python
from ask_sdk_core.skill_builder import SkillBuilder
from ask_sdk_core.dispatch_components import AbstractRequestHandler
from ask_sdk_core.dispatch_components import AbstractExceptionHandler
from ask_sdk_core.handler_input import HandlerInput

from ask_sdk_model import Response

class LaunchRequestHandler(AbstractRequestHandler):
    """Handler for Skill Launch."""
    def can_handle(self, handler_input):
        # type: (HandlerInput) -> bool

        return ask_utils.is_request_type("LaunchRequest")(handler_input)

    def handle(self, handler_input):
        # type: (HandlerInput) -> Response
        speak_output = "Bienvenido a la aplicación de Obtención de datos, que dato desea consultar"

        return (
            handler_input.response_builder
                .speak(speak_output)
                .ask(speak_output)
                .response
        )

sb = SkillBuilder()

sb.add_request_handler(LaunchRequestHandler())
sb.add_request_handler(GetDataIntentHandler())
sb.add_request_handler(LastQueryIntentHandler())
sb.add_request_handler(HelpIntentHandler())
sb.add_request_handler(CancelOrStopIntentHandler())
sb.add_request_handler(SessionEndedRequestHandler())
sb.add_request_handler(IntentReflectorHandler()) # make sure IntentReflectorHandler is last so it doe

sb.add_exception_handler(CatchAllExceptionHandler())

lambda_handler = sb.lambda_handler()
```

Figura 4-16: Ejemplo de controlador de intención en una Skill de Alexa

El controlador de cada Intent será una clase que extenderá la clase "AbstractRequestHandler", implementando los métodos de "can_handle", en el que se indica la Intent que va a controlar la clase, y "handle", que contiene la funcionalidad del controlador para procesar la petición de entrada y devolver la respuesta al asistente. Todos los controladores se añaden a un objeto "Skillbuilder", que será el punto de entrada para la Skill, controlando y enrutando las peticiones para ser procesadas por su controlador correspondiente.

Un problema que se ha encontrado es que al consultar datos de la nariz de un mes completo mediante la Intent principal "get_data", ocurría un error por *timeout* ya que, como se indicó en el Apartado 4.1 Problemas, el tiempo de respuesta máximo del controlador es de 8 segundos, y para una consulta pesada de este tipo se superaba dicho límite. Una primera solución consistía en implementar una Progressive Response de la API. Una respuesta de este tipo permite notificar al usuario de que la consulta es lenta y que se está ejecutando, evitando de esta forma que hay un momento de silencio antes de la respuesta. Sin embargo, se observó que esta funcionalidad no eliminaba la restricción de los 8 segundos, sino que simplemente servía para evitar que haya un momento de silencio y que la conversación con el asistente sea más agradable.

Aquí es donde entra en juego la intención extra "get_last_query" que se mencionó anteriormente. Una segunda solución propuesta fue ejecutar la consulta de datos en un

proceso o hilo separado y guardar el resultado en un archivo de texto almacenado en el servicio de almacenamiento S3, devolviendo al usuario una respuesta parcial notificando que la consulta se está ejecutando en segundo plano, y que puede obtener los resultados de su última consulta con el comando de voz "*Consultar último resultado*". A pesar de conseguir implementar esta funcionalidad, se encontró el problema de que tras devolver el controlador de la intención una respuesta al usuario, el proceso terminaba junto a todos los hilos o procesos hijo lanzados desde el mismo, por lo cual resultaba imposible ejecutar la consulta en segundo plano.

La solución definitiva fue crear un servicio en local con Flask y desplegarlo. En caso de que la intención principal "get_data" detecte que la consulta corresponde a una consulta lenta, como puede ser consultar los datos de un mes completo, en lugar de ejecutar la petición dentro del controlador, se realiza una petición a nuestro servicio local, indicando la consulta que se quiere realizar. Cuando nuestro servicio local recibe dicha petición, ejecuta la consulta y guarda su resultado en un fichero, almacenado localmente. Cuando el usuario llama a la intención "get_last_query" mediante el comando de voz "*Consultar último resultado*", el controlador de dicha intención manda una petición a nuestro servicio local para obtener la última consulta, el servicio local lee la consulta del fichero y devuelve el resultado. De esta forma se consigue evitar la restricción de los 8 segundo, con la desventaja de que el usuario debe realizar un comando de voz extra.

4.3.2.2 Rutina de alarmas

En Alexa no existe un límite de notificaciones como en Google, por lo cual podemos implementar el sistema de notificaciones mediante la API de Proactive Events. Para poder utilizar los eventos proactivos, es necesario añadir a nuestra Skill el permiso de poder enviar notificaciones al usuario. Cuando un usuario instale nuestra Skill, se le requerirá una confirmación de que permite recibir notificaciones de nuestra Skill.

Para enviar una notificación, la rutina de alarmas debe completar dos pasos:

- Obtener un token de autenticación de nuestra Skill. Para ello se realiza una petición HTTP a la API bajo la URL: *https://api.amazon.com/auth/O2/token* . En el cuerpo de la petición se indica el ID y la clave secreta de nuestra Skill.

- Una vez obtenido el token, se realiza otra petición, en este caso a la URL: *https://api.eu.amazonalexa.com/v1/proactiveEvents/stages/development* . Dicha URL corresponde a la API de eventos proactivos. En la cabecera de la petición se indica el token obtenido en la petición anterior, y en el cuerpo se debe enviar un objeto JSON que siga el formato de alguno de los eventos proactivos disponibles y que contenga el texto de la alarma que se desea enviar.

Un problema de las notificaciones de Alexa es que al tener que seguir el formato de algún evento proactivo de los definidos y disponibles en la API, no es posible enviar una respuesta completamente personalizada. Por ejemplo, uno de los eventos disponibles en la API es el *AMAZON.WeatherAlert.Activated* . Las notificaciones de dicho evento siguen el siguiente esquema: "*There is a <weatherAlert.alertType> alert for your area. Provided by <weatherAlert.source>.*", pudiendo personalizarse solamente los parámetros de tipo de alerta y fuente de la alerta. Por suerte, existe el esquema *AMAZON.MessageAlert.Activated* con el cual se pueden obtener alarmas muy parecidas a las que se enviarían si se pudiera personalizarlas: *"You have <messageGroup.count> <state.freshness> <state.status>*

message/messages from <messageGroup.creator.name>.". Los parámetros de *freshness* y *status* se pueden omitir, y en el parámetro de *creator.name* se puede insertar el nombre de nuestra Skill y texto de la alarma, quedando una alarma como: "*Tiene un mensaje de Obtención datos: Alarma, se ha superado el umbral…*".

4.4 Implementación de la consulta de datos de la nariz

Toda la funcionalidad de consulta y manejo de datos se ha implementado en una librería Python separada, de forma que las rutinas de alarma y comunicación con el asistente solamente tienen que llamar una función de la librería pasando por parámetros las características de la consulta.

La librería recibe como parámetros: el sensor a consultar, la característica que se quiere consultar, el rango temporal y el valor numérico del umbral en caso de ser necesario. La siguiente tabla especifica los valores que pueden tomar los diferentes parámetros (se han escogido tres sensores de la nariz para las pruebas de validación):

Parámetro	Valores
Sensor	CO_2 (co2_digital), aminas (tgs2603), humedad (bme280_humidity)
Característica	Umbral o máximos
Rango temporal	Último dato, ayer, semana pasada, mes actual, mes específico(enero-diciembre) del año actual
Umbral	Cualquier número

Tabla 4-1: Parámetros de consulta

Con los parámetros anteriores, se construye una petición GET y se envía a la URL indicada en el Apartado 3.3. Con los datos obtenidos, se realiza un procesado simple que consiste en recorrer los datos con una ventana de 1 hora o 1 día, con función de si el rango temporal es grande o pequeño. Para cada ventana se guarda el valor máximo registrado, o en caso de que la consulta especifique un umbral, se guarda el máximo si supera dicho umbral.

Una vez obtenidos los datos que se presentarán al usuario, se construye el texto correspondiente, que será devuelvo a la rutina que realizó la llamada. El texto está preparado para su envío directo al asistente.

En la actualidad, el usuario puede consultar cualquier combinación de parámetros listados anteriormente. Sin embargo, las rutinas de alarmas simplemente consultan los datos para los posibles sensores, emitiendo una alarma en caso de que uno o varios sensores superen el umbral especificado al lanzar las rutinas.

5 Integración, pruebas y resultados

5.1 Integración

La integración de la funcionalidad implementada es sencilla con el diseño propuesto. Para integrar las acciones personalizadas creadas para el asistente de Google, simplemente será necesario iniciar sesión con la cuenta con la que se han desarrollado las acciones en el dispositivo. De esta forma, al iniciar sesión en el asistente de Google en un teléfono móvil, se dispondrá automáticamente de la funcionalidad implementada. En el caso de Alexa, será necesario iniciar sesión en la aplicación de Alexa y en "Skills y juegos > Mis Skills" podremos activar nuestra Skill.

Para integrar la funcionalidad en un dispositivo como un altavoz inteligente, también bastará con iniciar sesión con la cuenta correspondiente, y ya tendremos acceso a nuestra herramienta.

Para que todo funcione, será necesario desplegar las rutinas de back end descritas anteriormente, siendo la única restrictiva la de alarmas para el asistente de Google, al tener que encontrarse el dispositivo que ejecuta la rutina conectado a la misma red que el asistente personal.

5.2 Pruebas

Las pruebas realizadas se han llevado a cabo tanto en las herramientas de prueba facilitadas por las herramientas de desarrollo Dialogflow y Alexa Developer Console, como con dos dispositivos físicos: un Google Home Mini para comprobar la funcionalidad del asistente de Google, y un Echo Dot para Alexa.

Las combinaciones de parámetros para los comandos de voz son múltiples, en el siguiente apartado 5.3 Resultados se expondrán ejemplos para cada posible comando.

5.3 Resultados

Este apartado del documento muestra el correcto funcionamiento de la funcionalidad en las herramientas de prueba de desarrollo, ya que de esta forma se presenta un chat físico en el que se pueden escribir los comandos deseados y aparecen las respuestas en modo de texto. No es posible demostrar el funcionamiento de las alarmas de esta misma forma, ya que éstas llegan directamente al dispositivo, por ello para comprobar el funcionamiento de las alarmas se han de consultar los vídeos de los dispositivos físicos de los enlaces listados en la Tabla 5-1.

Nombre del archivo	Descripción	Enlace
Alexa_Obtencion_Datos_Conversacional.mp4	Ejemplo de conversación con un Echo Dot en el que se ejecutan varios comandos de voz	Link
Alexa_Ejemplo_Notificacion.mp4	Ejemplo del funcionamiento de las alarmas con Alexa	Link
Google_Assistant_Obtencion_Datos_Conversacional.mp4	Ejemplo de conversación con un Google Home Mini en el que se ejecutan varios comandos de voz	Link
Google_Assistant_Ejemplo_Notificacion.mp4	Ejemplo de funcionamiento de las alarmas con Google	Link

Tabla 5-1: Descripción y enlaces a demostraciones del funcionamiento de los asistentes con dispositivos físicos

Para conseguir los resultados de los ejemplos de rutinas de alarmas, se forzó a las rutinas a enviar una alarma, indicando un valor bajo para un sensor en concreto. En un funcionamiento normal, la rutina se encuentra en continuo funcionamiento, consultando datos cada 2 minutos y lanzando una alarma solamente en el caso de que se supere algún umbral y no se haya lanzado ninguna alarma en los últimos 30 minutos.

A continuación, se muestran los resultados obtenidos en las consolas de desarrollo, al realizar diferentes comandos variando los diferentes posibles parámetros. Para no sobrecargar el documento, no se incluyen todas las posibles combinaciones, pero si las suficientes para ilustrar el correcto funcionamiento de la funcionalidad implementada.

5.3.1 Resultados obtenidos para el asistente de Google

Lo primero que debemos hacer es invocar nuestra acción personalizada, con un comando como "*Hablar con obtención datos*". Como se puede comprobar en la Figura 5-1, tras ejecutar el comando el asistente nos da la bienvenida a la aplicación.

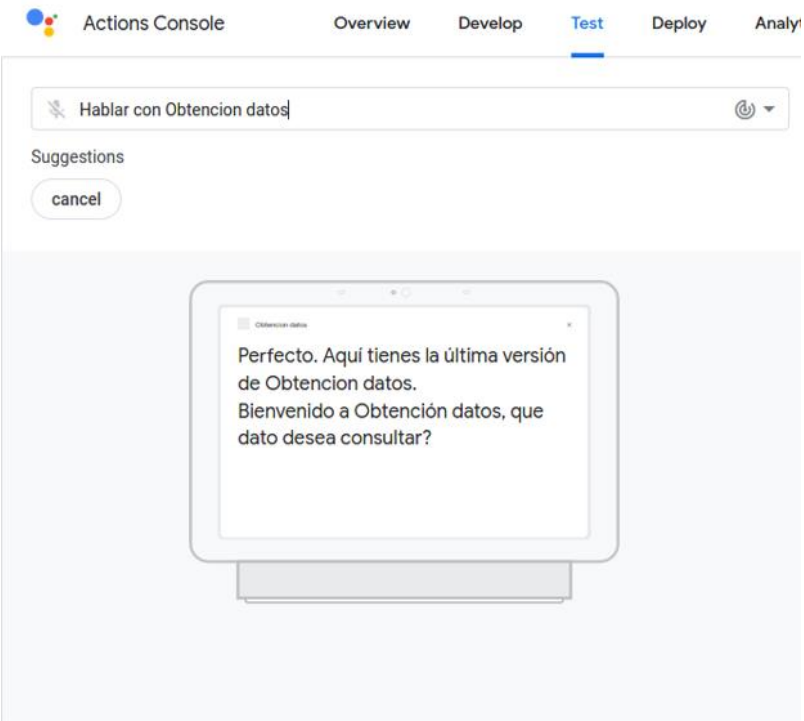

Figura 5-1: Comando inicial Google Assistant

A continuación, probamos el comando más sencillo, que consiste en obtener el último valor registrado de un sensor. La Figura 5-2 muestra la respuesta del asistente a dicho comando para los diferentes sensores.

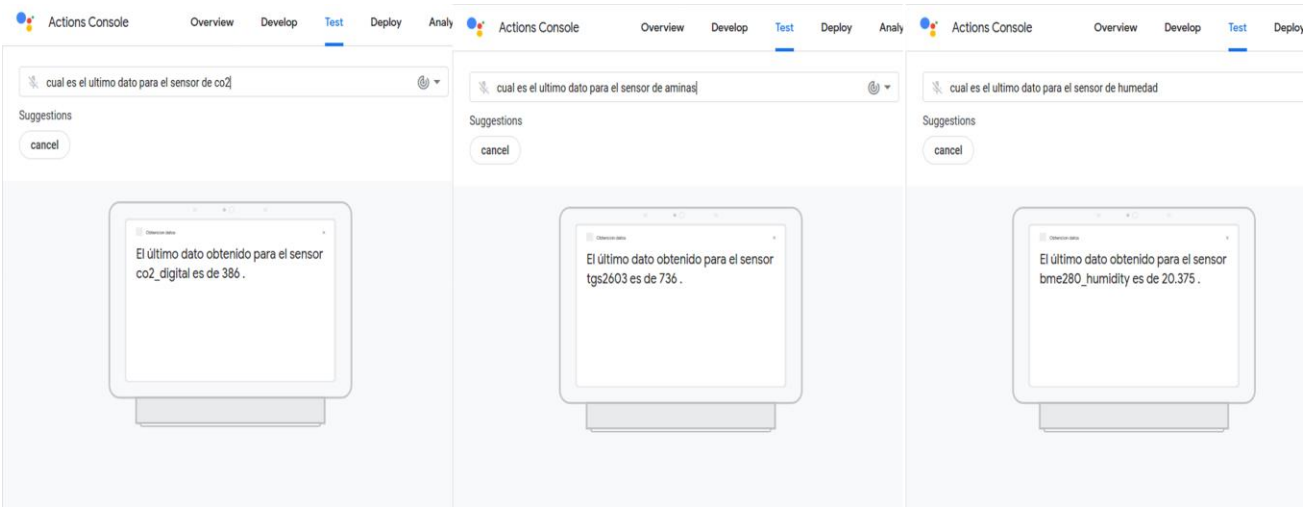

Figura 5-2: Comando consulta de último dato

A continuación, probamos comandos que obtienen los valores máximos de cierto sensor para diferentes periodos de tiempo. La Figura 5-3 ilustra la respuesta del asistente a comandos para obtener valores máximos del día de ayer o de la semana pasada. Para consultas que tienen un periodo de tiempo pequeño, como es el caso de un día, se proporciona un máximo valor del sensor por cada hora del día, proporcionándose 24

valores en caso de la consulta de datos de un día; para consultas que corresponden a un periodo de tiempo mayor, como puede ser una semana o un mes, se proporciona el mayor dato de cada 24 horas del periodo.

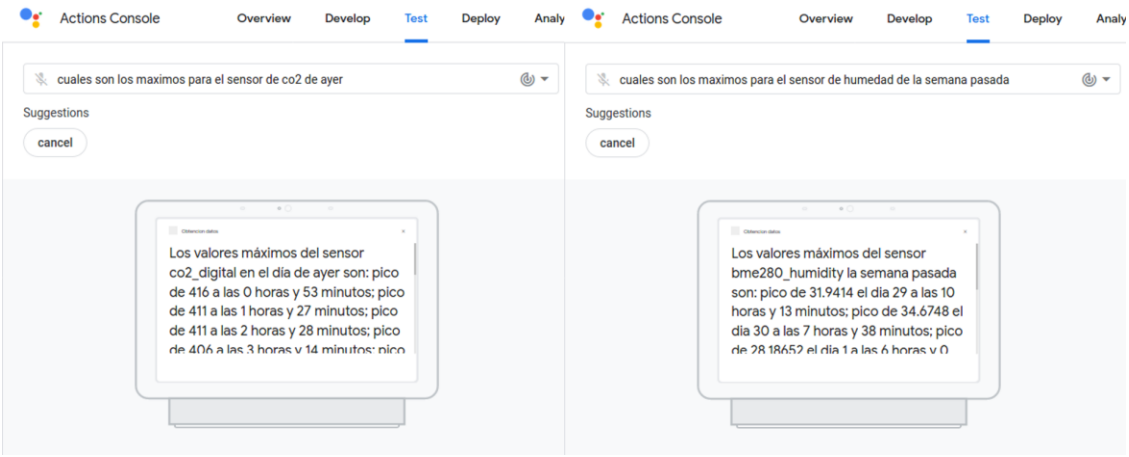

Figura 5-3: Consulta de máximos del día de ayer y de la semana pasada

Es posible consultar los datos del mes actual, o de un mes en concreto del año actual, como ilustra la Figura 5-4. Sólo se pueden consultar datos de meses pasados, en caso de consultar un mes futuro se obtendrá un mensaje de error.

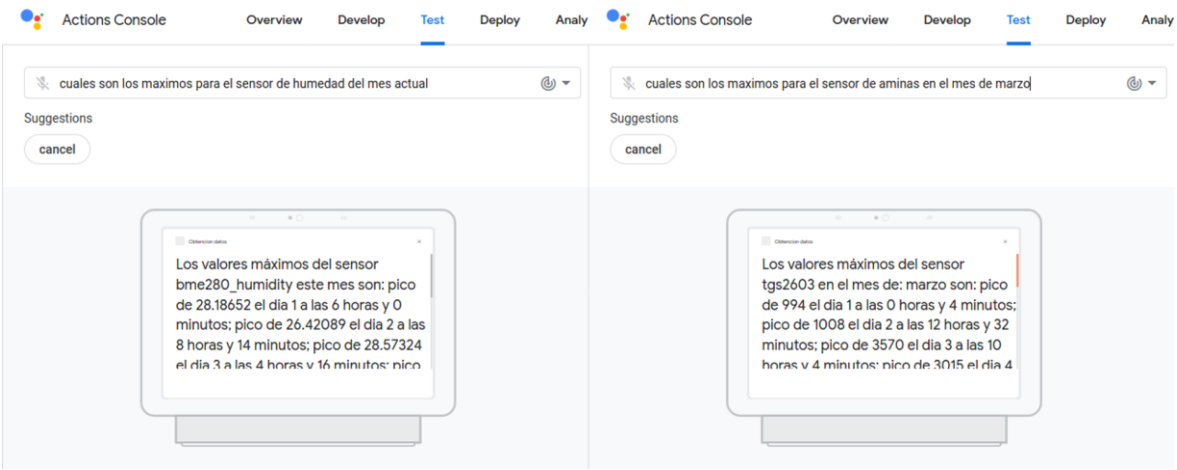

Figura 5-4: Consulta de valores máximos del mes actual y un mes específico

Finalmente, se muestran los resultados de ejecutar comandos que obtienen los valores máximos, pero sólo los que superen cierto umbral. Como se puede observar en la Figura 5-5, todos los valores presentados por el asistente son superiores al umbral especificado en el comando, y en caso de que no haya ningún valor que supere el umbral, se mostrará el mensaje correspondiente.

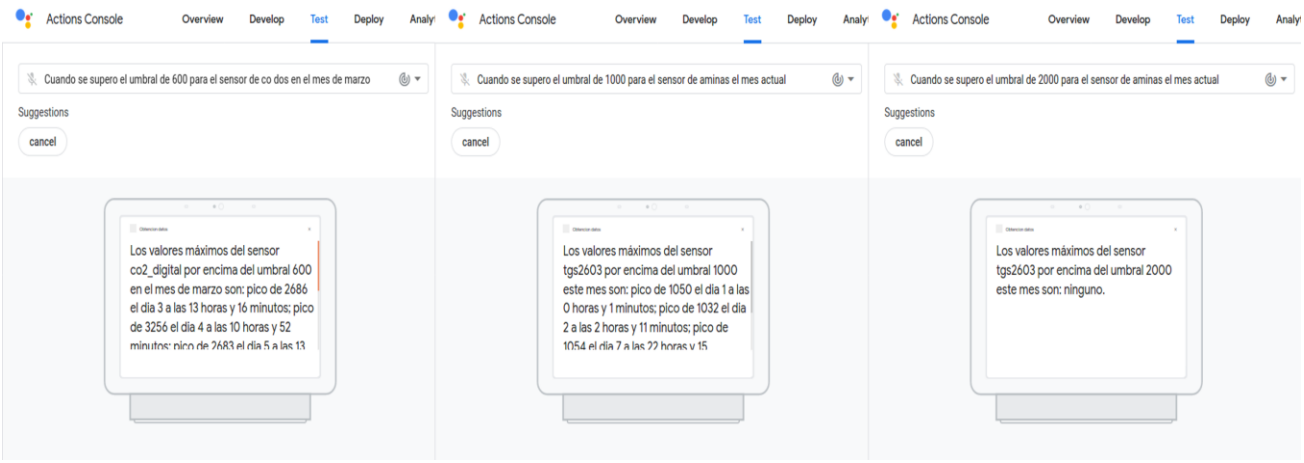

Figura 5-5: Consulta de valores por encima de un umbral

5.3.2 Resultados obtenidos para Alexa

Los resultados obtenidos para el asistente de Alexa son muy similares a los ya descritos para el asistente de Google, por ello no se hará mucho hincapié en describirlos, pero se añaden para que se pueda observar que la funcionalidad desarrollada funciona igual de bien que la desarrollada para el asistente de Google.

Al igual que con el asistente de Google, primero es necesario decirle a Alexa que queremos ejecutar nuestra Skill, esto se hace como se puede observar en la Figura 5-6.

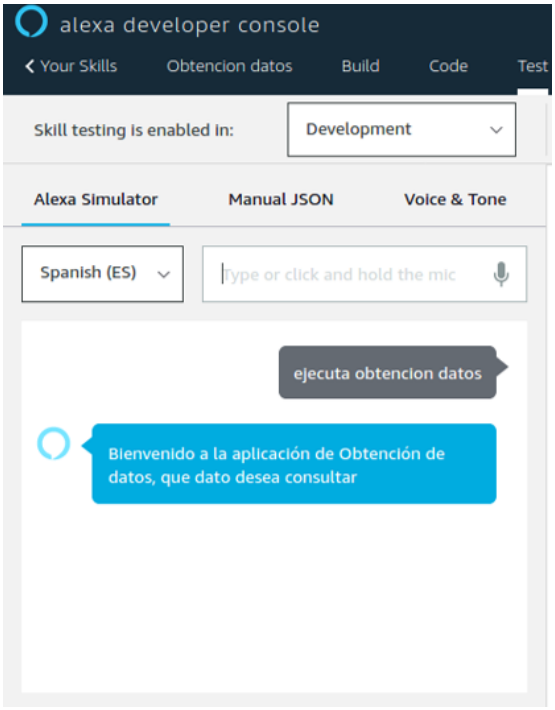

Figura 5-6: Invocación Skill de Alexa

La Figura 5-7 muestra ejecución de comandos para consultar el último dato de un sensor.

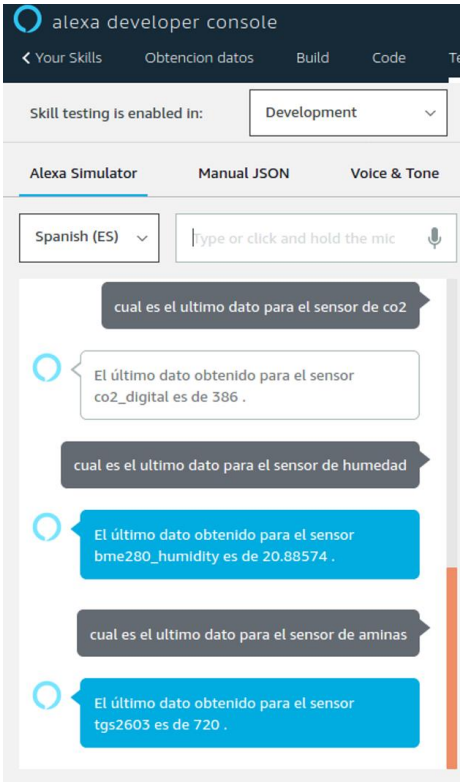

Figura 5-7: Consultas de último dato de cada sensor en Alexa

En la Figura 5-8 se pueden observar las consultas de valores máximos para periodos pequeños de tiempo, devolviéndose el resultado directamente. En la Figura 5-9 por el contrario, se observan consultas para periodos más amplios de tiempo, por lo cual, como se ha explicado en apartados anteriores, es necesario que el usuario primero ejecute su consulta deseada, y posteriormente ejecute una consulta adicional para recuperar el valor de la consulta anterior.

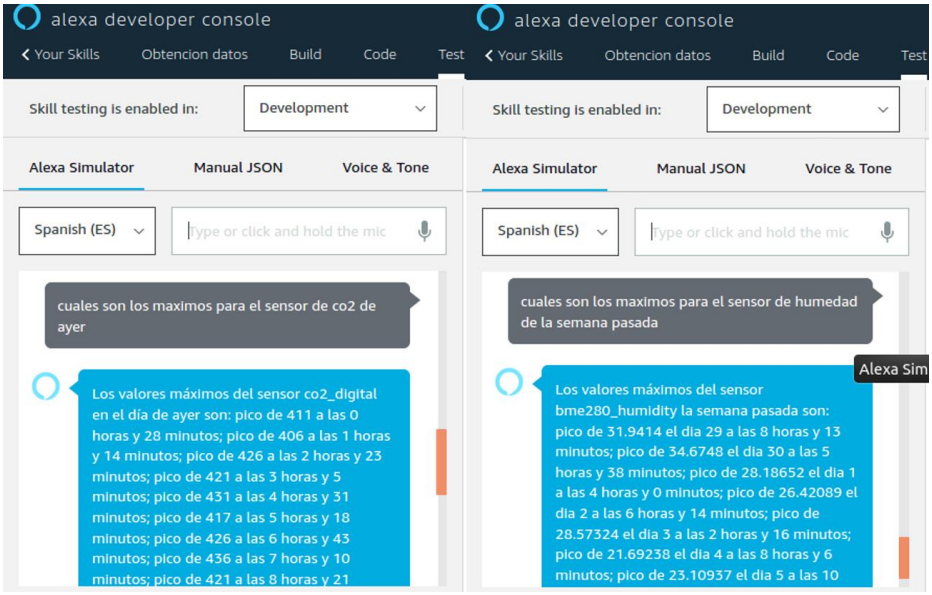

Figura 5-8: Consulta de máximos en Alexa

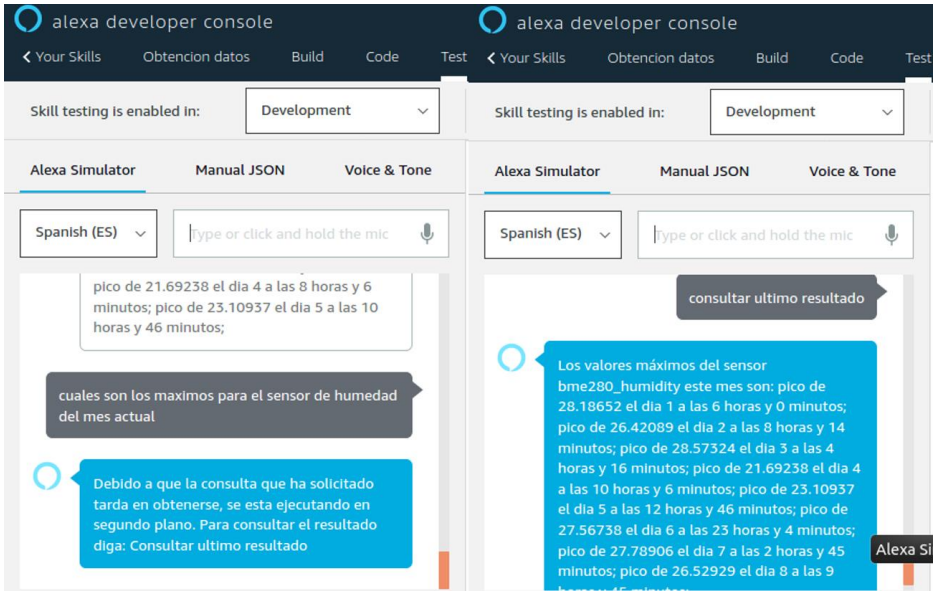

Figura 5-9: Consulta de máximos para periodos de tiempo grandes en Alexa

La necesidad de ejecutar una consulta en dos comandos también se aplica a los comandos que obtienen máximos por encima de cierto umbral, ya que, aunque es posible que la cantidad de datos final sea pequeña, es necesario realizar una consulta al almacenamiento de datos para el periodo completo. Dicho comportamiento se ilustra en la Figura 5-10, siguiendo las consultas un orden de izquierda a la derecha.

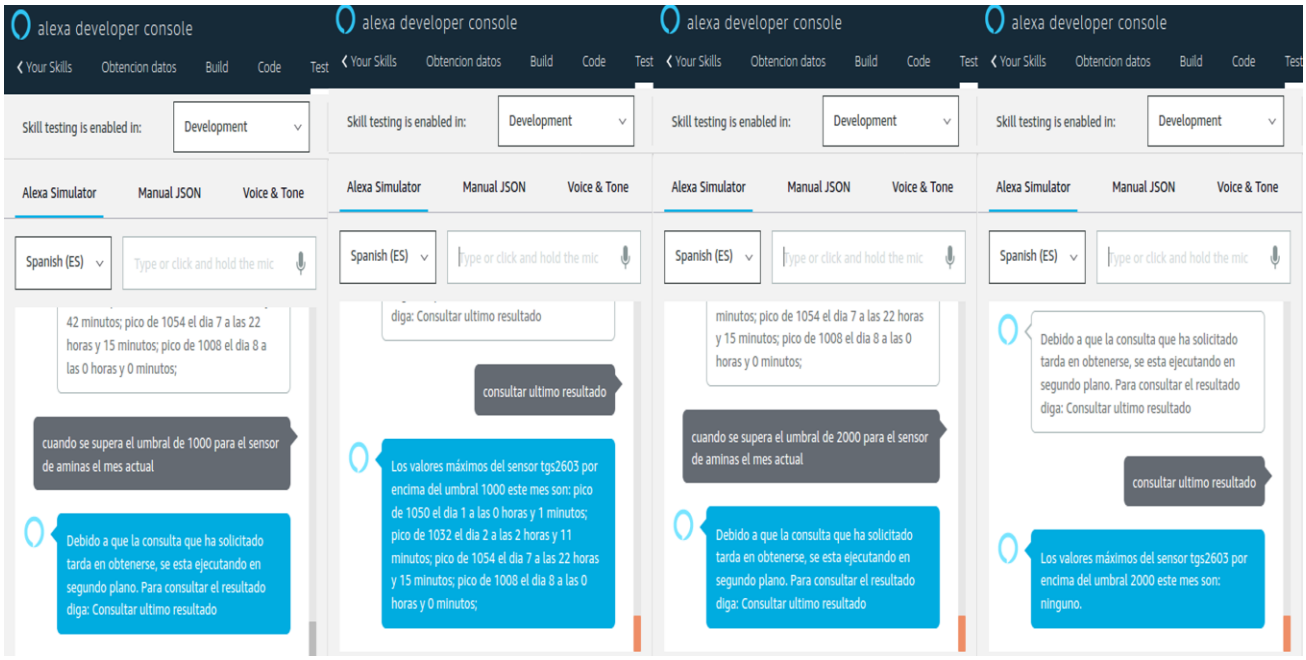

Figura 5-10: Consulta de máximos por encima de cierto umbral en Alexa

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Como se ha podido observar con los resultados aportados, la funcionalidad implementada cumple con los requisitos funcionales y permite al usuario interactuar con los asistentes de Google y Alexa mediante comandos de voz. Dichos comandos de voz permiten especificar diferentes parámetros como el sensor a consultar, el rango temporal, si se quieren consultar valores máximos o valores máximos por encima de cierto umbral, y el umbral en caso de ser necesario. La respuesta obtenida está acorde a dichos parámetros. La interacción con los asistentes resulta cómoda e intuitiva, siendo los tiempos de respuesta lo suficientemente bajos para que la conversación sea lo más cercana posible a la humana.

Además de la comunicación directa con el asistente, se ha implementado un sistema que permite recibir alarmas en asistentes a partir de los datos obtenidos a lo largo del tiempo con una nariz artificial en una tarea de monitorización no invasiva de rutinas humanas. En el trabajo desarrollado, se expone un ejemplo de alarma en el que cuando se detecta un dato de algún sensor que supere el umbral establecido, el asistente avisará del evento indicando sensor y valor detectado.

En cuanto al manejo de los datos provenientes de la nariz artificial, se ha desarrollado un algoritmo simple que procesa los datos obteniendo valores máximos con ventanas de diferente tamaño en función del rango temporal especificado en los comandos de voz. El sistema soporta diferentes rangos temporales, como el día de ayer, la semana pasada, el mes actual o un mes específico del año.

Cabe destacar la escasa información en relación con las APIs de los asistentes y la sorprendente falta de flexibilidad a la hora de manejar algunos aspectos como los tiempos de respuesta límites o los límites de envío de notificaciones. Es esperable que a medida que

se desarrolle la tecnología y surjan nuevas aplicaciones como la propuesta en este trabajo, se proporcione mayor flexibilidad para los desarrolladores.

A pesar de los múltiples problemas encontrados debido a ciertas restricciones que presentan los asistentes personales a la hora de desarrollar funcionalidad personalizada, se han encontrado e implementado soluciones alternativas que no empeoran la experiencia del usuario y que consiguen mantener el comportamiento deseado.

Se han cumplido los objetivos principales del trabajo, obteniendo como resultado una potente herramienta que puede ser utilizada en múltiples ámbitos. Se ha demostrado el correcto funcionamiento de la funcionalidad desarrollada tanto con simulaciones como con dispositivos físicos, que dan una idea del comportamiento de la herramienta en un entorno real.

6.2 Trabajo futuro

La herramienta que ha sido desarrollada forma una funcionalidad base, que puede ser extensible.

Algunos aspectos claros donde se puede realizar más desarrollo son: añadir más comandos de voz al asistente y realizar diferentes análisis de los datos de la nariz con algoritmos de detección de eventos más complejos que los utilizados para ilustrar la funcionalidad de la interacción entre sensores olfativos y asistentes personales. Ambos aspectos van de la mano, ya que para integrar uno es necesario también integrar el otro.

Una mejora muy interesante sería automatizar diferentes procedimientos que incluyan dispositivos adicionales como el control de la ventilación automático en función de las alarmas.

Otro aspecto que puede ser mejorado es el servicio web de back end implementado para el asistente de Google. Las peticiones que envía el asistente a dicho servicio llevan datos de quién está realizando las peticiones, pudiendo de esta forma comprobar si alguien ajeno está intentando acceder a los datos, y añadiendo así una capa de seguridad al sistema. Dichas comprobaciones, en caso del back end de Alexa, se realizan de forma automática, al tener el servicio desplegado en un servidor de Amazon y hacer uso de la API *ASK SDK*.

Es previsible que las APIs de los asistentes personales tanto de Google como de Amazon vayan incorporando nuevas funcionalidades y protocolos que permitan aumentar el tipo y la variedad de interacciones con dispositivos multisensor. Esto permitirá desarrollar nuevas aplicaciones para usos diversos en el contexto de las tecnologías del Internet de las Cosas.

Referencias

[1] K. Persaud, G. H. Dodd, Analysis of discrimination mechanisms of the mammalian olfactory system using a model nose, Nature, Vol. 299, 1982. https://doi.org/10.1038/299352a0

[2] Karakaya, D., Ulucan, O. & Turkan, M. Electronic Nose and its Applications: A Survey. Int. J. Autom. Comput. 17, 179–209 (2020). https://doi.org/10.1007/s11633-019-1212-9

[3] Vergara, A., Muezzinoglu, M. K., Rulkov, N., and Huerta, R. (2010). Information-theoretic optimization of chemical sensors. Sensors Actuators B Chem. 148, 298–306. https://doi.org/10.1016/j.snb.2010.04.040

[4] Vembu, S., Vergara, A., Muezzinoglu, M. K., and Huerta, R. (2012). On time series features and kernels for machine olfaction. Sensors Actuators B Chem. 174, 535–546. https://doi.org/10.1016/j.snb.2012.06.070

[5] Vanarse, A., Espinosa-Ramos, J. I., Osseiran, A., Rassau, A., & Kasabov, N. (2020). Application of a Brain-Inspired Spiking Neural Network Architecture to Odor Data Classification. Sensors, 20(10), 2756. https://doi.org/10.3390/s20102756

[6] Walt, D. R., Stitzel, S. E., and Aernecke, M. J. (2012). Artificial noses. Am. Sci. 100, 38–45. https://doi.org/10.1146/annurev-bioeng-071910-124633

[7] Gòmez A.H., Hu G., Wang J., Pereira A.G. Evaluation of tomato maturity by electronic nose. Computers Electr. Agric. 2006;54:44–52. https://doi.org/10.1016/j.compag.2006.07.002

[8] https://www.figaro.co.jp/en/technicalinfo/principle/mos-type.html

[9] Fernando Herrero-Carrón, David J. Yánez, Francisco de Borja Rodríguez, Pablo Varona, An active, inverse temperature modulation strategy for single sensorodorant classification, Sensors and Actuators B: Chemical, Volume 206, January 2015, Pages 555-563. https://doi.org/10.1016/j.snb.2014.09.085

[10] Liu, T., Li, D., Chen, Y., Wu, M., Yang, T., & Cao, J. (2020). Online Drift Compensation by Adaptive Active Learning on Mixed Kernel for Electronic Noses. Sensors and Actuators B: Chemical, 128065. https://doi.org/10.1016/j.snb.2020.128065

[11] Global smart speaker market share 2018 and 2019 by platform, Statista Research Department, Feb 19, 2020

[12] McLean, G., & Osei-Frimpong, K. (2019). Hey Alexa… examine the variables influencing the use of artificial intelligent in-home voice assistants. Computers in Human Behavior, 99, 28-37. https://doi.org/10.1016/j.chb.2019.05.009

[13] https://console.actions.google.com/

[14] https://developer.amazon.com/alexa/console/ask

[15] https://pypi.org/project/PyChromecast/

[16] https://pypi.org/project/gTTS/

Glosario

UAM	Universidad Autónoma de Madrid
API	Application Programming Interface
NLP	Natural Language Processing
MBUX	Mercedes-Benz User Experience
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
URL	Uniform Resource Locator
Framework	Entorno de trabajo
AWS	Amazon Web Services
Timeout	Error por superación de un tiempo de espera máximo establecido
ASK SDK	Alexa Skills Kit Software Development Kit

Anexos

A Manual de instalación

En este apartado del anexo se detallan los pasos necesarios para desplegar todos los componentes que hacen posible la comunicación en el diseño propuesto.

Para los asistentes de Google y Alexa, simplemente será necesario iniciar sesión con la cuenta con la que se ha desarrollado las habilidades de los asistentes en el dispositivo que se quiere utilizar. Si se quiere utilizar la funcionalidad con otra cuenta, será necesario publicar nuestra Action o Skill. Otra opción, sería crear la misma funcionalidad para la cuenta nueva siguiendo los pasos descritos en el Apartado 4-2 Implementación de las características del asistente.

Para la parte del servidor, será necesario desplegar la aplicación web en la ruta indicada en el apartado de Fulfillment, como se puede ver en la Figura 4-7. Nuestro servicio web para Alexa se encuentra siempre desplegado, sin embargo, será necesario desplegar la rutina que realiza consultas lentas en local.

Para las rutinas de alarmas, en el caso de Alexa simplemente será necesario ejecutar la rutina pasando por parámetro el umbral de cada sensor. Para la rutina de alarmas implementada para el asistente de Google será necesario ejecutar la rutina pasando por parámetro el nombre del dispositivo* y los umbrales. Además de lo anterior, será necesario iniciar un simple servidor HTTP en el directorio donde se guardan los archivos .mp3 creados con gTTS, esto se puede hacer fácilmente con el siguiente comando:
"*python -m http.server"*.

*Dicho nombre se puede encontrar en la aplicación de Google Home, una vez enlazamos nuestro dispositivo con la aplicación.